OMI-569

Further Development of a Computational Vision Model for the Generation and Analysis of Photo-Realistic Scenes for Military and Civilian Virtual Reality Applications

Final Report (CDRL A002) and Software User's Manual (CDRL A004)

Gary Witus

February 1996

Prepared for:

U. S. Army Tank-automotive Command
Warren, MI 48397-5000

Under Contract Number:

DAAE07-95-C-R071

OptiMetrics, Inc.
3115 Professional Drive
Ann Arbor, Michigan 48104-5131

TABLE OF CONTENTS

1.0 INTRODUCTION 1
1.1 RESEARCH OBJECTIVES 1
1.2 RESEARCH ACTIVITIES AND RESULTS 1
1.2.1 SYNTHETIC IMAGERY 2
1.2.2 FUNCTIONAL REQUIREMENTS FOR CVM EARLY VISUAL PROCESSING FORMULATIONS 2
1.2.3 "LATE" VISION PROCESSING MODEL WITH MULTIPLE-HYPOTHESES 3

2.0 ALGORITHMS AND MODELS (ANALYSTS' MANUAL) 5
2.1 SYNTHETIC IMAGE GENERATION 5
2.1.1 NON-STATIONARY MULTI-RESOLUTION FILTERING FOR "NO TARGET CONTRAST" 5
2.1.2 SYNTHETIC TEXTURE 8
2.1.3 BLENDING TO CREATE LOW-SIGNATURE STIMULI FOR PERCEPTION TESTING 13
2.2 "EARLY" VISUAL PROCESSING MODEL FOR IMAGE ANALYSIS 15
2.2.1 LUMINANCE ADAPTATION AND NON-LINEAR CONE RESPONSE 15
2.2.1.1 PUPIL-TO-RETINA STEADY-STATE RESPONSE 16
2.2.1.2 LUMINANCE-ADAPTED RECEPTOR INSTANTANEOUS RESPONSE 19
2.2.1.3 APPLICATION TO COMPUTATION VISION MODELING 22
2.2.2 SHAPE FILTERING 23
2.3 "LATE" VISION PROCESSING MULTIPLE-HYPOTHESIS MODEL FOR SIGNATURE METRICS 24

3.0 SOFTWARE USER'S MANUAL 34
3.1 INSTALLATION 34
3.1.1 RETRIEVAL FROM TAPE 34
3.1.2 ADDING OR UPDATING A USER 34
3.2 OPERATION 35
3.2.1 TURN-KEY OPERATION 35
3.2.1.1 SELECT AND CONVERT AN IMAGE FOR ANALYSIS. 35
3.2.1.2. SPECIFY THE SEARCH REGION 36
3.2.1.3 SPECIFY THE TARGET REGION 36
3.2.1.4 EVALUATE A SIMPLE METRIC AND ESTIMATE THE PROBABILITY OF DETECTION 37
3.2.2 INTERFACE CONFIGURATION AND HIGH-LEVEL DATA FLOW CONTROL 38
3.2.2.1 GENERATE MASKS AND REFERENCE IMAGES 38
3.2.2.2 COMPUTE MEANLUMINANCE AND EYENOISEPOWER 40
3.2.2.3 SELECT MASK AND PYRAMID FILES TO USE TO COMPUTE METRICS 41
3.2.2.4 SET ADAPTATION PARAMETERS (OPTIONAL) 41
3.2.3 ANALYZE THE PYRAMIDS 42
3.2.4 COMPUTE METRICS AND PROBABILITIES 42
3.2.5 POST_PROCESSING UTILITIES 43

3.2.6 DYNAMIC SCENE ANALYSIS 44
3.3 MODIFICATION 45
3.4 KNOWN ERRORS AND PLANNED IMPROVEMENTS 48
3.4.1 KNOWN ERRORS 48
3.4.2 PLANNED IMPROVEMENTS 49

APPENDIX A - REFERENCES
APPENDIX B - CURRENTLY AVAILABLE GENERAL PURPOSE MODULES
APPENDIX C - APPLICATION-SPECIFIC MODULES IN NAC VPM

LIST OF ILLUSTRATIONS

Figure 2-1. Enlarged example of "notarget contrast" processing. 7
Figure 2-2. Algebraic specification for non-stationary spatial filtering extrapolation. 8
Figure 2-3. Data flow for non-stationary spatial filtering extrapolation algorithm as implemented on the workbench. 9
Figure 2-4. Synthetic texture data flow diagram. 10
Figure 2-5A. Original scene denoting regions to receive synthetic texture. 11
Figure 2-5B. Scene with synthetic texture inserted. 12
Figure 2-6. Evidence of shape-element neural filters (from Thompson, 1967). 14
Figure 2-7. Steady state cone response as a function of uniform luminance input (derived function and numerical approximation). 19
Figure 2-8. Instantaneous cone response as a function of luminance input at five different adaptation levels, assuming no summation due to receptor coupling. 21
Figure 2-9. Instantaneous cone response as a function of luminance input at five different adaptation levels, with half-constant for summation due to receptor coupling at 16 cd/m^2. 22
Figure 2-10. NAC VPM dynamic processing flow. 26
Figure 2-11. NAC VPM static processing flow. 27
Figure 2-12. Generate masks and pyramids. 27
Figure 2-13. Compute mean luminance. 28
Figure 2-14. Compute eye noise power vector. 28
Figure 2-15. Select pyramid and mask pyramid. 28
Figure 2-16. Compute adaptive gains and transformed power pyramids. 29
Figure 2-17. Compute metrics and probabilities with alternative aggregation methods. 29
Figure 2-18. Parameters in the "Adaptive Gain Parameters" Input file, and their use in VPM processing. 30
Figure 2-19. Predicted probability image computation. 31
Figure 2-20. Metric and probability A computation. 31
Figure 2-21. Metric B. 32
Figure 2-22. Metric and probability C1. 32
Figure 2-23. Metric and probability C2. 33
Figure 3-1. Example data flow diagram and corresponding vision simulation workbench script. 46
Figure 3-2. Vision simulation workbench formal grammar. 47

1.0 INTRODUCTION

This document and the accompanying software constitute CDRL A002, Final Report, CDRL A003, Computer Software, and CDRL A004, Software User's Manual, for Small Business Innovative Research Phase I contract number DAAE07-95-C-R071, entitled "Further Development of a Computational Vision Model for the Generation and Analysis of Photo-Realistic Scenes for Military and Civilian Applications."

This introduction summarizes the research objectives, methods and results. Section 2 presents the detailed research results and model/algorithm formulations. Section 2 constitutes the analysts' manual for the software. Section 3 contains the software user's manual.

1.1 RESEARCH OBJECTIVES

The objective of the Phase I effort was to investigate the application of computational vision models (CVM) to simulate human performance in image analysis tasks, military target acquisition, automotive collision avoidance, and visual virtual reality for perception testing. The specific objectives are listed below:

(1) To develop, implement and assess algorithms and software to generate synthetic imagery with photo-realistic appearance of the natural surrounding image, for application to create low signature stimuli for visual virtual reality for perception testing;

(2) To define functional requirements to enhance and refine the computational vision model of the "early" stages of visual processing (from entrance at the pupil through receptive field linear spatial filtering) implemented in the TARDEC National Automotive Center Visual Performance Model (VPM); and

(3) To develop and implement a model of alternative hypotheses regarding "late" vision processing stages (i.e., after the "early" stages of visual processing, up to cognitive decision and response) for the purpose of formulating alternative signature metrics to predict decision and response behavior.

1.2 RESEARCH ACTIVITIES AND RESULTS

The research was conducted, building on the foundation of the CVM theory and algorithms in the VPM, and was implemented in software using the TARDEC National Automotive Center Vision Simulation Workbench for rapid-prototyping visual processing simulation software. The research activities consisted of developing mathematical models, evaluating them empirically, and comparing them to findings in the literature of vision research.

1.2.1 SYNTHETIC IMAGERY

Synthetic image generation algorithms were developed to create the appearance of the surrounding image in a specified target region. The synthetic images provide insight into what visual characteristics the VPM early visual processing model is and is not sensitive to. They also provide a means to create low signature stimuli for visual virtual reality for perception testing.

The synthetic image generation procedure is organized into three sequential algorithms. The first algorithm modifies the content of the target region to minimize luminance and chromatic contrasts between the target region and its surround, at all points on the target/surround boundary on all multi-resolution spatial frequency planes. The second algorithm adds a synthetic, random texture, on each multi-resolution spatial filtering plane, to the interior of the target. The third algorithm blends the original target with a synthetic image, to achieve any linear combination of the two on each of the multi-resolution planes.

1.2.2 FUNCTIONAL REQUIREMENTS FOR CVM EARLY VISUAL PROCESSING FORMULATIONS

The synthetic image samples were examined to identify deficiencies in the early visual processing formulation. In general, the synthetic texture algorithm was not photo-realistical in its reproduction of surrounding textures. The synthetic texture was photorealistic when surrounded by "shapeless" textures such as deciduous treetop foliage, but was not photo-realistic when surrounded by strongly "shaped" textures, such as long grass or tree trunks.

Further investigation showed that the problem is that the simple shape used in the VPM early visual processing model (a short, stubby line segment at two orientations) is inadequate to characterize natural textures for photo-realistic synthetic imagery. The synthetic texture appearance was not significantly different for the natural textures when the two oriented shape filters were replaced with a single shape: an unoriented spot. The quality of the photorealism was improved when the 3-by-3 convolution kernel used in the shape filtering was replaced with a 5-by-5 kernel, thus extending the correlation length or equivalently reducing the orientation bandwidth from 90 to 45 degrees.

The vision research literature also supports the conclusion that a richer set of shape filters is needed. Neurophysiological studies have found that the neural receptive fields responding to oriented line segments have orientation bandwidths ranging from 5 to 360 degrees, with a median of 45 degrees. Other neurophysiological studies have found receptive fields responding to more complex shapes such as corners. Recent studies in search have demonstrated early visual processing filters for many local shape elements, and that this filtering is a significant factor in search time and target conspicuity.

In keeping with the goal of providing an extendible research system, a simple approach to restructure the VPM software to extend the set of shape element filters was formulated.

The process of summarizing laboratory display requirements and tradeoffs for visual virtual reality led to the investigation of early adaptation and non-linearities in the visual system. These processes are not currently modeled in the VPM early visual processing model. A mathematical formulation of luminance adaptation and cone non-linearities suitable for incorporation into the VPM was developed. This formulation will support future investigation into the impact of luminance compression on stimuli realism in laboratory settings.

1.2.3 "LATE" VISION PROCESSING MODEL WITH MULTIPLE-HYPOTHESES

Research was conducted to formulate alternative hypotheses regarding how to model the performance of non-linear stages of visual processing following receptive field linear spatial filtering and leading to cognitive decision and response behavior. The vision researcher using the model can direct the processing flow via a number of software "switches" or "connections" representing different hypotheses regarding how the multi-resolution bandpass image representation is subsequently processed. There are six switches, which can be set to different values in any combination, thus producing a large number of distinctly different signature metrics. The software switches implement the following processing choices:

1. Measure the target signature from the early vision analysis (EVA) pyramid created from (a) original image including target and background, or (b) the difference between the original image and the "no target contrast" image with contrast modulation due to the target suppressed, i.e., the image of the difference due to the target.

2. Measure the target signature and background noise based on (a) contrast modulation, or (b) texture (contrast modulation magnitude) modulation.

3. Normalize the signal to (a) internal eye noise only, or (b) internal eye noise and scene clutter surrounding the target.

4. Apply adaptive gains representing the effects of training and familiarity: (a) to do nothing, i.e., no adaptive gains, (b) to reject the background, i.e., based on background characteristics only, (c) to select the target, i.e., based on target characteristics only, or (d) to reject the background and select the target.

5. Measure the signature as: (a) the cumulative normalized signal strength by integrating the normalized signal at multi-resolution pixels, or (b) the expected number of "outlier" multi-resolution pixels by integrating the probability of detecting a multi-resolution pixel (using a non-linear cumulative distribution function applied to the normalized signal).

6. Measure the signature by integrating over: (a) the target area, (b) the target/surround perimeter, (c) internal and external target edges, or (d) internal target region, i.e., the target area minus the target/surround perimeter.

This model is intended to be used for research in the formulation of predictive models of visual task performance. The multiple signature metrics provide an initial basis for screening and evaluation leading to robust predictive models. Different signature metrics may be appropriate for different visual tasks and context. There may be different ways in which observers can process the image in their task, in which case several different signature metrics may need to be used in combination. For complex targets, consisting of distinct components or segments, components may need to be evaluated individually, and the aggregate target perceptibility may need to be computed from the perceptibility of the components. Graph structures representing the logical relationships among the components may be appropriate model structures. The scope of Phase I did not include any screening to assess which metric or combinations of metrics, and how they should be applied, to predict decision and response for different visual tasks.

2.0 ALGORITHMS AND MODELS (ANALYSTS' MANUAL)

2.1 SYNTHETIC IMAGE GENERATION

2.1.1 NON-STATIONARY MULTI-RESOLUTION FILTERING FOR "NO TARGET CONTRAST"

An algorithm was developed and implemented using non-stationary multi-resolution spatial filtering to extrapolate the surrounding RGB pixel values into the target region. This procedure creates a image such that the content of the target region does not add modulation to the image: i.e., the image pyramid produced by stationary Gaussian pyramid lowpass filtering is identical to that produced by non-stationary Gaussian pyramid lowpass filtering in which the content of the target pyramid is not used. On each multi-resolution plane, it replaces the modulation influenced by the target region with the modulation induced solely by the surrounding scene, without using any image information from inside the target region. The contrast modulation from the target region is minimized in the sense that the apparent contrast modulation in the image is the same as if the observer's visual receptive fields were surgically modified so that they received no input from the target region.

The "no target contrast" image serves as a baseline for determining the modulation in the image due to contrasts created by the target. The only contrast modulation in the target region of the "no target contrast" is that induced by the surrounding scene, and the surrounding scene is identical to that of the original image. Therefore, the contrast induced by the target is simply the difference of the two images. The contrast modulation on each visual channel due to the target is simply the difference in the multi-resolution spatial filter pyramids of the original and "no target contrast" images.

The "no target contrast" image also serves as the first step in generating the appearance of a synthetic camouflage for a particular position in a scene. The RGB pixel values inside the target region of the "no target image" are chosen so that they add no contrast modulation across the target/surround boundary at any multi-resolution plane. However, the "no target" image is smoothed or blurred on the interior. It does not add contrast gradients, but it generally does create texture gradients. Synthetic texture needs to be added to achieve photo-realistic appearance of the surround.

Figure 2-1 shows an enlargement of an example of this processing. The target region is the rectangle in the center of the image straddling the boundary between the foreground grass and background treeline. The synthetic content of the target region converges to the local surround at each point on the boundary. It preserves the low frequency modulation from light at the bottom to dark at the top. It is locally bright next to small bright patches in the surround. Contrasts across the boundary at every position and scale are simultaneously minimized. However, the boundary remains readily apparent because of the difference in texture between the target region and the surround.

The algorithm to minimize luminance and chromatic contrasts across the target/surround boundary performed to expectations. At all multi-resolution planes, at all receptive field locations overlapping the target/surround boundary, the modulation in the synthetically modified image was the same as if the receptive fields had been altered so that they did not receive input from the target region. The surround image is unaffected. The synthetic image minimizes the contrast modulation in the sense that the modulation with the synthetic target image is no different than that which would have been detected had the receptive fields "missed" the target location, i.e., the synthetic image did not add contrast modulation to the image.

The concept of non-stationary spatial filtering is similar to stationary spatial filtering using convolution. Convolution is the basic operation in the multi-resolution lowpass spatial filtering in VPM. For a stationary filtering process, the convolution kernel is constant regardless of where it is being applied in the image. In non-stationary filtering, the kernel changes with position in the image. To create the effect of the visual system not receiving input from the target region, the convolution kernel is modified as follows:

1 - when the kernel maps onto pixels that all lie outside the target region, the kernel is unchanged;

2 - when some of the kernel cells map inside the target region and some map outside, the values of kernel cells that map onto the target region are set to zero, and remaining cells in the kernel are scaled up by the same factor such that they sum to unity; and

3 - when all of the kernel maps into the target region, the convolution value is undefined, and is arbitrarily set to zero.

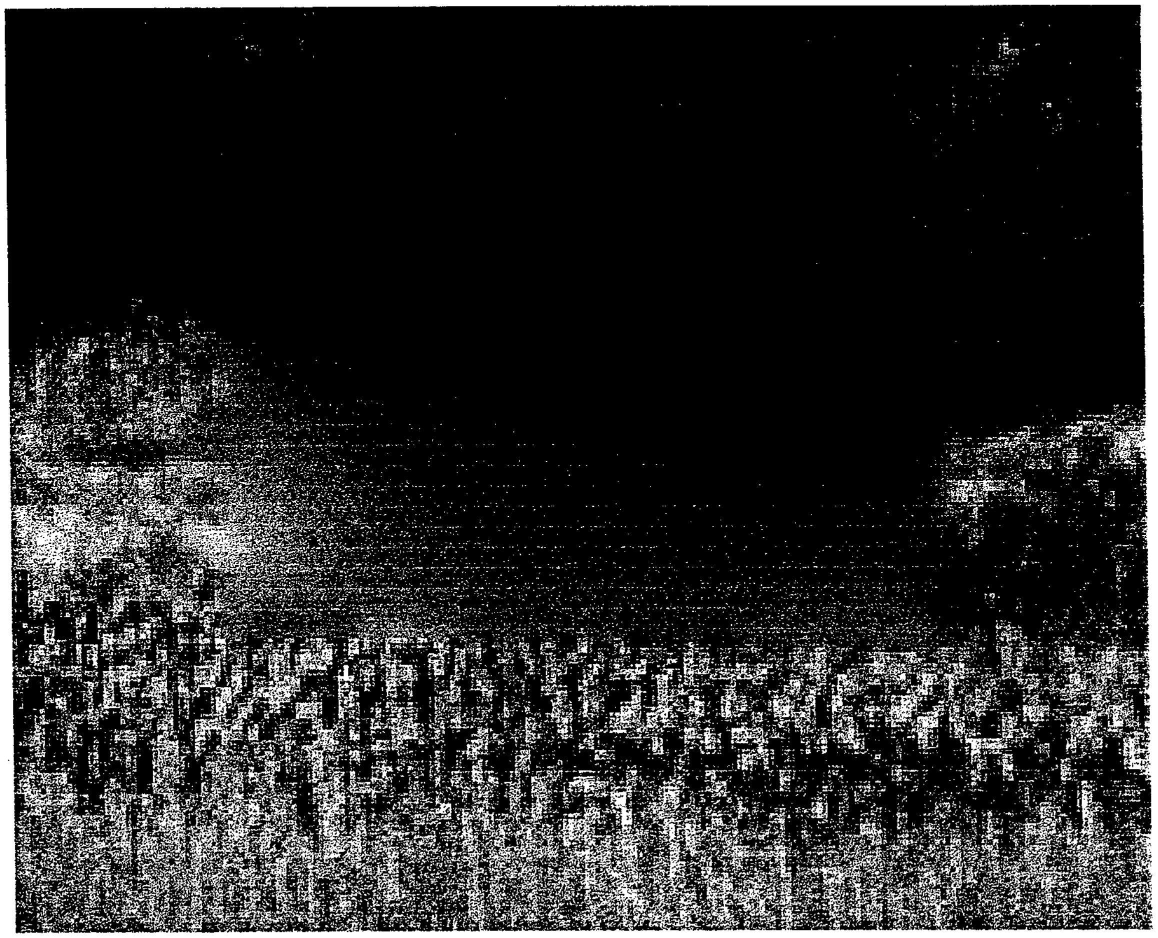


Figure 2-1. Enlarged example of "no target contrast" processing.

Implementing the non-stationary filtering by changing the kernel with spatial position is hopelessly inefficient. However, an efficient algorithm to compute the same result was devised. The algorithm is presented in Figure 2-2. The data flow diagrams of the implementation on the workbench are shown in Figure 2-3.

2.1.2 SYNTHETIC TEXTURE

An algorithm was developed and implemented using stationary multi-resolution spatial filtering to add synthetic texture to a region. Texture is measured by the amplitude envelope of the Early Vision Analyzer (EVA) spatial bandpass filter image pyramid. The synthetic texture is random in phase and has amplitude equal to the average amplitude surrounding the target region. This operation is applied to the target region in the "no target" image to create a synthetic camouflage pattern. The objective is to obscure the texture gradient across the boundary between the target and its surround.

I	=	Image Input
M	=	Target Mask Input
L	=	Stationary lowpass-and-subsample spatial filter operator
I'	=	No Target Image Output
Y_i	=	Gaussian Pyramid Plane i of $I \bullet M$
W_i	=	Gaussian Pyramid Plane i of M
X_i and Z_i	=	Temporary Pyramids
imax	=	Lowest Frequency (Highest Number) Plane
$Y_{\varnothing}$	=	$I \bullet M$

$$Y_{i+1} = \ell(Y_i)$$

$$W_{\varnothing} = M$$

$$W_{i+1} = \ell(W_i)$$

$$X_i = \left(Y_i \Big/ \left(W_i + \varepsilon^2\right)\right) \cdot \begin{Bmatrix} 1 & \text{when } W_f(i,j) > \varepsilon \\ 0 & \text{else} \end{Bmatrix}$$

$$Z_{imax} = X_{imax}$$

$$Z_i = X_i + \ell^{-1}(Z_{i+1}) \cdot \begin{Bmatrix} 1 & \text{when } X_f < \varepsilon \\ 0 & \text{else} \end{Bmatrix}$$

$$I' = Z_{\varnothing}$$

Figure 2-2. Algebraic specification for non-stationary spatial filtering extrapolation.

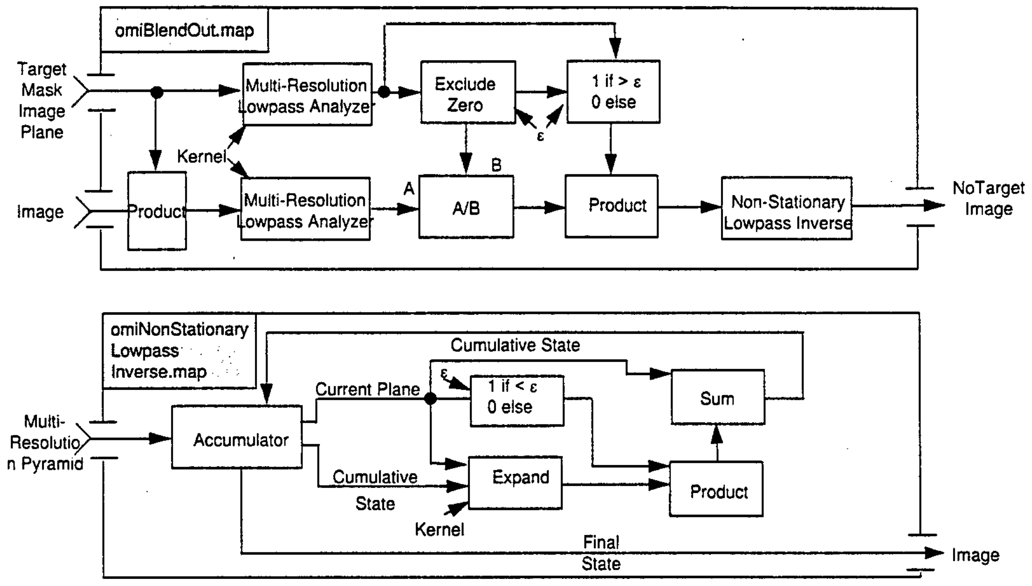


Figure 2-3. Data flow for non-stationary spatial filtering extrapolation algorithm as implemented on the workbench.

The synthetic texture algorithm computes the average texture in the surround immediately outside of the target region on all multi-resolution planes, and adds a random texture to the multi-resolution interior of the target region. Only the modulation at receptive field locations which are strictly interior to the target region are modified, those which overlap the target and surround are unchanged. This ensures that the operations do not distort the background. The synthetic texture is random in phase. The amplitude has a normal distribution with zero mean and magnitude equal to the average magnitude in the immediate exterior of the target. The data flow diagram for the algorithm is presented in Figure 2-4.

The purpose of the synthetic texture generation was to explore the adequacy of the spatial filtering operations in the early visual processing model for recreating photo-realistic textures. The synthetic texture algorithm was intended only for cases in which the target was surrounded by a single non-heterogeneous texture. When the target region is on the boundary between two different textures in the surround, the average texture will not necessarily resemble either texture. Furthermore, if the two adjacent textures in the surround also have different luminances, contrast at the boundary will create additional texture which will be included in the average texture. This case can be handled by applying the non-stationary extrapolation algorithm to the texture pyramid of the "no target contrast image," provided that the texture representation is adequate. Therefore, the research focused on assessing the adequacy of the texture measure.

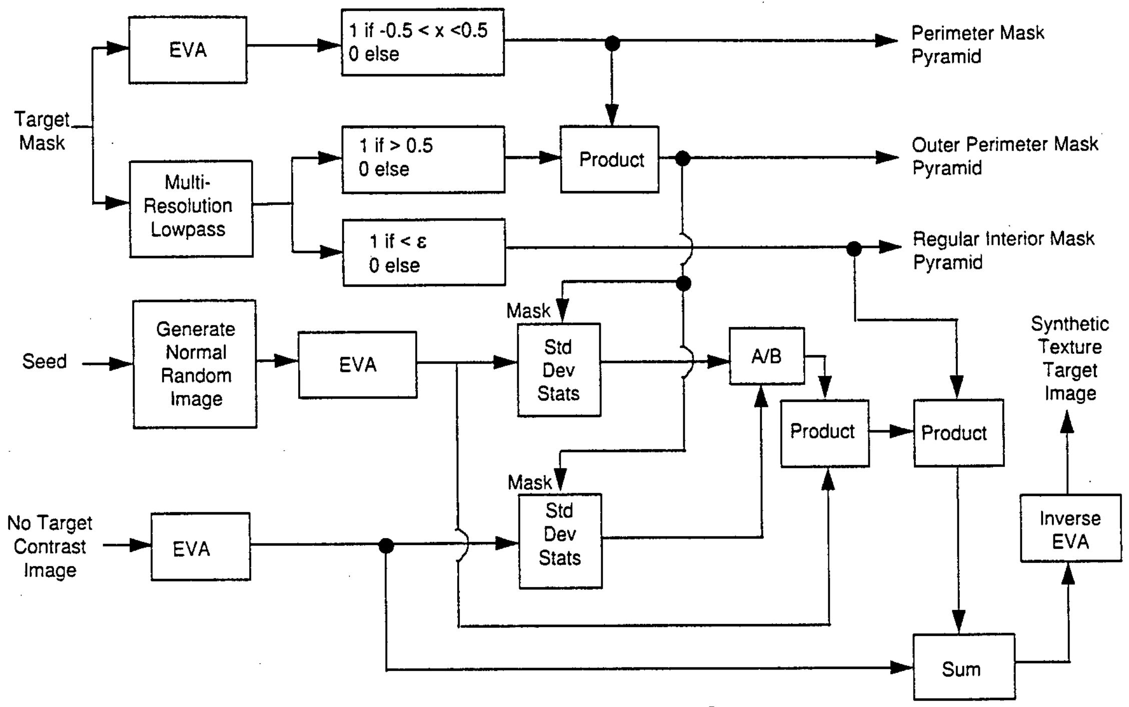


Figure 2-4. Synthetic texture data flow diagram.

Figure 2-5 illustrates the results of this algorithm. The contents of the square in the middle of the grass, the square in the middle of the tree leaves, and the square on the treeline are all synthetic textures. The example shows that the algorithm does not adequately recreate the visual characteristics of even homogeneous textures. The texture of the treetops is unoriented, and the algorithm works fairly well for the rectangle in the treetops, as expected. Also, as expected, it does not work well for the target rectangle on the treeline boundary. What was suprising was that the synthetic grass texture was not well matched in the rectangle in the grass.

The texture of the grass has a very pronounced vertical orientation, obviously lacking in the synthetic texture of the rectangle in the grass. Detailed examination of the grass texture shows that the correlation lengths are considerable longer than can be represented in a 3-by-3 orientation filtering kernel in the EVA module, or equivalently, the grass orientation bandwidth is considerably narrower than the 90-degree bandwidth of the EVA module. Further testing showed that when the orientation filtering was bypassed, the appearance of the synthetic grass was indistinguishable

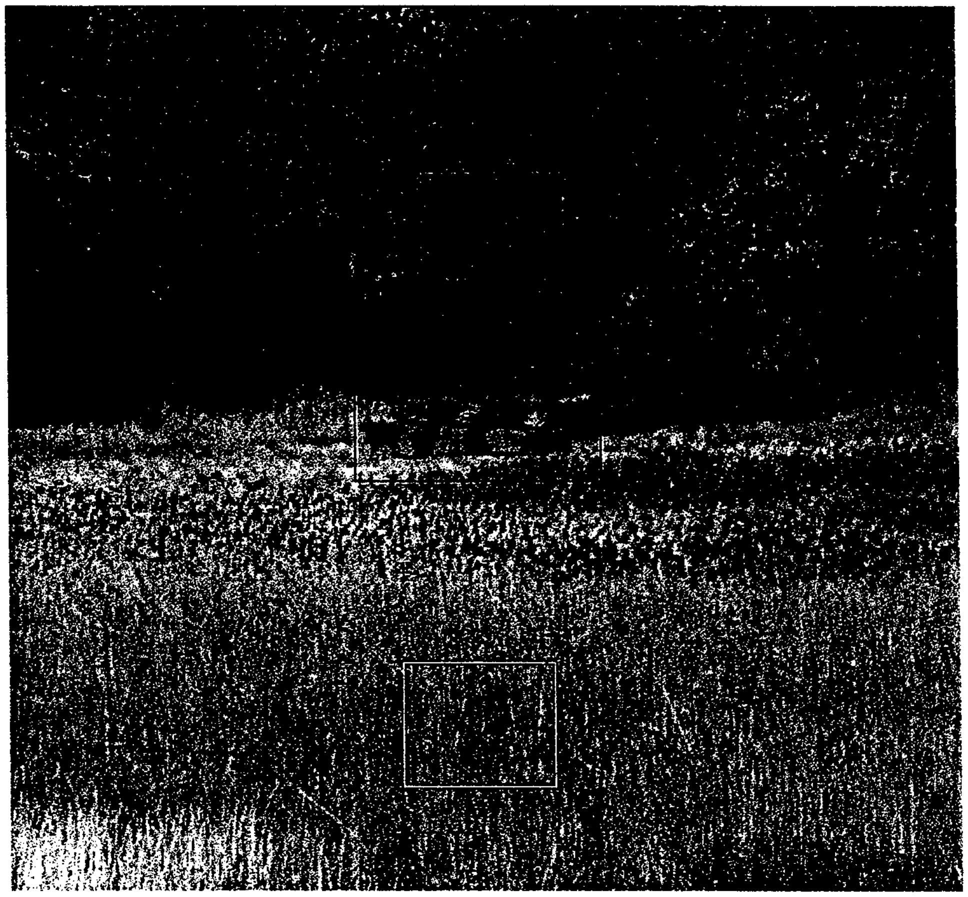


Figure 2-5A. Original scene denoting regions to receive synthetic texture.

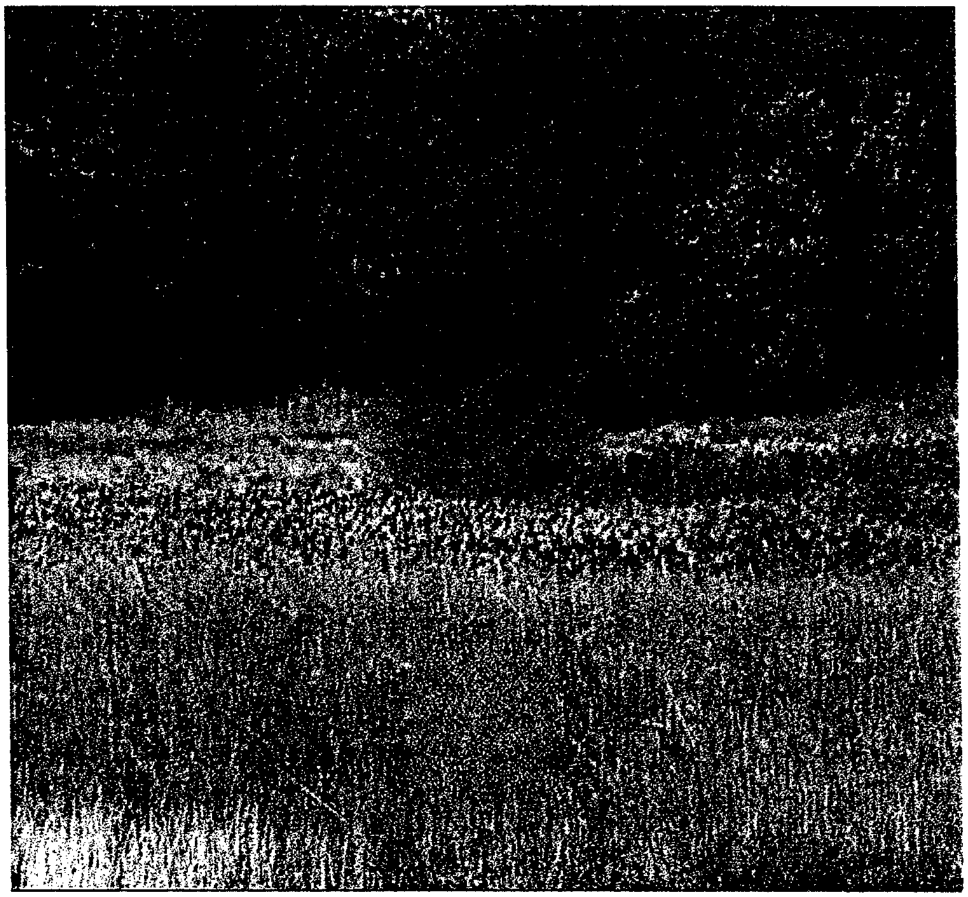


Figure 2-5B. Scene with synthetic texture inserted.

from that created with the two orientation filters. Only when artificial patterns were used did the difference become apparent. In summary, the shape-element filtering in the EVA module which represents only one shape element, a short stubby line segment at two orientations, is inadequate to reproduce the texture of nature foliage with photo-realism.

This demonstrates that a better representation of shape elements in the EVA is needed. This conclusion is also substantiated by additional vision research findings regarding shape element processing in early visual processing, and its role in search. Figure 2-6 illustrates the basic research finding "corner detector" receptive fields in the cat. Anecdotally, corners are a feature which contribute to the detection of military targets with otherwise low signature. Other basic research (DeValois and DeValois, 1990) shows that receptive fields tuned to line-segment shapes occur with orientation bandwidths from 5 to 360 degrees.

At the 1995 Conference of the Association of Vision Researchers and Ophthalmologists, Dr. Jeremy Wolfe from the Harvard Medical School presented the results of his research which shows that shape-element filtering in the early visual processing is a significant factor in search. The term "shape element" refers to a local structure, a portion of a closed loop, but not the outline of an entire object. In Dr. Wolfe's research he used a small set of primitive shape elements at different orientations to construct a target with a unique global shape and a set of distractor objects. When the target included a unique shape element not present in the distractors, the time to find the target was independent of the number of distractors.

When the target had a unique global shape but no single unique shape element, the time to find the target was linear in the number of distractors. (Dr. Wolfe has not postulated a canonical set of shape elements for human vision.)

2.1.3 BLENDING TO CREATE LOW-SIGNATURE STIMULI FOR PERCEPTION TESTING

Low signature stimuli for perception testing can be created by blending the original image with either the "no target contrast" image or with the camouflaged target image created by adding synthetic texture to the "no target contrast" image. Since the images differ only inside the target region, only the target appearance is altered.

A simple approach is simply to use a linear combination of the two images, e.g.,

New_Image = α * Original_Image + (1 - α) * Camouflage_Image,

where α is a constant between zero and one.

This method only allows the researcher one degree of freedom. It does not allow the researcher control of the trajectory between the original image and the camouflage image. All of the visual channels receive the same scaling factor. For the purpose of creating stimuli for perception testing, it is desirable to be able to control the linear combination on individual visual channels independently.

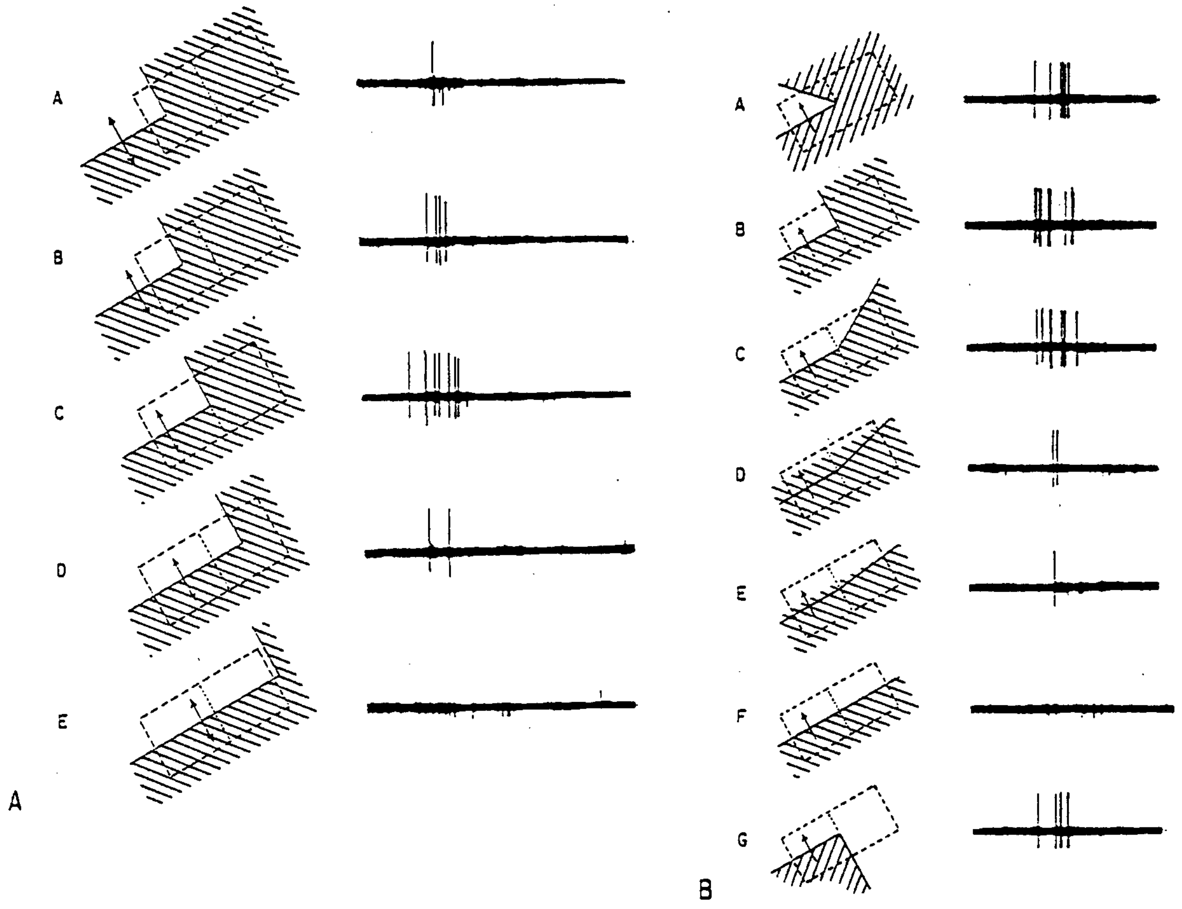


Figure 11.25. Examples of the behavior of a hypercomplex cell in cortical visual area II of the cat. As in Figure 11.23, the stimuli are shown on the left and the cell discharges on the right. This cell appears to function in general terms as a "right angle detector. " In A, responses of the cell are shown to the right angle stimulus moved across different regions of the field (A-E). In B, various stimulus angles are moved across the field (A-G). Angles closest to right angles produce the greatest response. (From Hubel and Wiesel, 1965)

Figure 2-6. Evidence of shape-element neural filters (from Thompson, 1967).

The third algorithm in the set to create low-signature stimuli computes the multi-resolution linear combination of two images, with independent scale factors on each visual channel. As implemented, however, the current algorithm is incomplete: the step was omitted which ensures that the background was unchanged and only the target region was changed. The current multi-resolution linear blending algorithm implements the following equation:

$$\text{New\_Image} = \text{Inv\_EVA}\,(\,\alpha_i\ *\ \text{EVA}_i\,(\text{Original\_Image}) + (\,1 - \alpha_i\,)\ *\ \text{EVA}_i\,(\text{Camouflage\_Image})\,).$$

where EVA_i (.) denotes the ith visual channel, and Inv_EVA denotes the inverse of the EVA processing.

The following step was omitted and needs to be added:

$$\text{Fixed\_New\_Image} = \text{New\_Image} - \text{NTC}(\text{New\_Image}) + \text{NTC}(\text{Original\_Image}),$$

where NTC(.) denotes the non-stationary lowpass filtering operation used to create the "no target contrast" image. This step was included in the TVM, but was omitted due to an oversight, in the blending utility provided with the VPM software.

2.2 "EARLY" VISUAL PROCESSING MODEL FOR IMAGE ANALYSIS

The Phase I research developed approaches to refine and extend the Early Vision Analysis formulation inherited from the baseline TARDEC Visual Model (TVM). These approaches were not implemented in Phase I and have been deferred to Phase II for implementation and testing. An improved formulation of luminance adaptation and non-linearities in the cone output was developed. Also developed was an approach to represent shape element filtering which occurs at the receptive-field stage of early visual processing. This method removes the restriction to line-segment shape elements. It employs an extensible set of user-defined shape element filters.

2.2.1 LUMINANCE ADAPTATION AND NON-LINEAR CONE RESPONSE

The baseline VPM formulation does not represent the non-linear aspect of cone response: it implicitly assumes that the output is a linear function of the input. The assumption is approximately true over any narrow range of luminance. However, it is not true over the dynamic range of luminance in a natural scene.

The approximation was originally made because it is common in the vision research literature examining contrast thresholds. By definition, contrast threshold studies operate over a small dynamic range.

The correction is to introduce a transfer function to represent the eye's adaptation to the mean luminance and the non-linear cone response, given the luminance adaptation. The importance of this transformation became apparent during the assessment of laboratory stimuli presentation relative to natural world conditions.

This section presents a mathematical analysis of luminance adaptation and receptor (cone) response non-linearities. It is extends Boynton's (1992) analysis of steady-state luminance adaptation and cone response non-linearity to address the instantaneous response of the luminance-adapted visual system, i.e., the response of individual receptors over a single fixation between saccades when viewing spatially and temporally non-uniform scenes. The analytic formulation is suitable to incorporate into simulations of visual processing to represent the effects of luminance adaptation and cone response non-linearities. The analytic formulation is applied to characterize the luminance transformations which, when applied to the visual input, leave the retinal output unaffected. This result has application to assessing the fidelity of perceived contrast in video display of natural scenes.

Effective visual discrimination occurs over an enormous range of light intensities. Cone vision operates over a range from 0.3 to 300,000 candela per square meter, or 6 decades. However at any given instant in time, the range is much more restricted, approximately 2 to 3 decades. Luminance adaptation is the process by which the relatively limited dynamic range of the instantaneous sensitivity is shifted to accommodate the level of illumination. Analytic formulations of both steady-state luminance adaptation and the instantaneous response of the luminance-adapted eye are needed to construct computational models of visual spatio-temporal pattern processing, and to assess the fidelity of perceived contrast in laboratory reproductions of natural scenes.

2.2.1.1 PUPIL-TO-RETINA STEADY-STATE RESPONSE

Boynton (1992, pp. 175-179) provides a concise summary and analysis of steady state cone output in response to the level of illumination entering the eye. Boynton notes that his formulation does not account for the observed linear response to increments and decrements around the steady-state value, nor does it account for the observed linear response to variations at low intensities. Boynton's model describes steady-state response, not the response of an individual cone in the context of a visual input to the entire eye, or the spatio-temporal pattern of retinal output of the luminance-adapted eye.

Boynton's steady-state analysis addresses the value to which cone output settles, in response to a sustained, uniform luminance input. The steady-state conditions are an acceptable approximation to natural situations provided that the average luminance does not vary significantly with time or with eye movement. In some natural situations, adaptation transients can temporarily affect vision (e.g., moving between outdoor clear sunlight and interior artificial illumination, turning on or off a bright light at night, or shifting gaze between bright clouds and a dark shadowed area on the ground). However, in many situations of interest, adaptation transients can be ignored, since the scene luminance does not change significantly with time or with eye position. Given these conditions, non-uniformities in the spatio-temporal pattern of light entering the eye will not cause the luminance adaptation to differ significantly from the steady-state adaptation because the mechanisms of the luminance adaptation involve spatio-temporal lowpass filter processes, i.e., they respond slowly relative to eye movements and movements in the scene, and they respond to the aggregate input over large areas relative to the area of an individual receptor.

Boynton identifies three mechanisms that transform the range of cone response: pupil reflex, photopigment bleaching/regeneration, and cone response non-linearity. Pupil reflex and photopigment bleaching/regeneration are both luminance adaptation mechanisms. Steady-state cone response non-linearity, as presented by Boynton, is the result of the combined effects of the inherent non-linearity of the receptors and interaction among neighboring receptors. This distinction is important to reformulating Boynton's model to separate luminance adaptation and the instantaneous response of the luminance-adapted eye (i.e., the spatio-temporal pattern of cone output in response to the spatio-temporal pattern of luminance input).

Boynton cites LeGrand (1968) for data on relative pupillary area as a function of stimulus luminance. Assuming a minimum pupil radius of 1 mm at 300,000 cd/m^2, the pupillary area in square millimeters can be computed simply by multiplying by pi. The following function for pupil radius, P in mm, as a function of luminance input, I in cd/m^2, fits LeGrand's data over the range of luminance input from 0.3 to 300,000 cd/m^2:

$$P(I) = 1 + 1.56 * \exp(-0.03 * (\ln(I) + 1.16)^2)). \qquad (1)$$

Pupil reflex is a response to the aggregate luminance input (given constant accommodation distance). No source of data or analysis of how the spatial pattern of luminance is aggregated to drive pupil input is known. An initial formulation would be to use the average luminance weighted by the cosine of the angle, i.e., weighted by the projected area of the pupil. Provided the scene luminance does not contain large luminance gradients over large spatial extents, the average luminance will not vary significantly with eye movements, and the steady-state approximation is applicable.

The aggregate retinal stimulation, S in trollands, is by definition equal to the product of the pupillary area in square mm and the average stimuli luminance in cd/m^2:

$$S(I) = pi * P(I)^2 * I. \qquad (2)$$

In natural viewing, the gaze shifts over the scene. After each saccade, each cone is exposed to a different level of stimulation than it was before the saccade. Since bleaching and regeneration are relatively slow compared to the mean dwell time between saccades (Boynton, 1992, pp. 169 to 174), the fraction of photopigment bleaching can be considered to be approximately the same for all the cones in the fovea, and the fraction can be estimated from the average stimulus intensity on the retina, i.e., from the steady-state response. The mean dwell time between saccades is only 1/8 second. The cone regeneration time constant is 120 seconds. The bleaching time constant is inversely proportional to the light intensity, but, on the basis of personal experience of adapting to outdoor illumination, is at least one second and possibly several seconds for strong daylight. Boynton uses the following equation for the fraction, F, of unbleached photopigment, with the half-bleach constant value of 20,000 trollands (after Rushton and Henry, 1968):

$$F(I) = S(I) / (S(I) + 20000). \qquad (3)$$

Note the form of this equation. It is the first-order rational polynomial approximation for detector response. It has the properties that it is linear for low input, saturates at high input, and is specified by a single parameter, the half-constant.

The energy collected by the cones, E in trollands, is equal to the product of the unbleached fraction of photopigment and the retinal stimulation in trollands:

$$E(I) = F(I) * S(I). \qquad (4)$$

Boynton cites his own research (Boynton and Whitten, 1970) for an equation for the steady-state receptor response, R in arbitrary units,

$$R(I) = E(I)^{0.7} / (E(I)^{0.7} + 631). \qquad (5)$$

The system of equations (1) through (5) describes the steady-state receptor output as a function of luminance input. Figure 2-7 illustrates the steady-state relationship by graphing the steady-state receptor response, R(I), as a function of the steady state input, I. Note that the x-axis is on a log scale. Figure 2-7 also includes a graph of the following numerical approximation:

$$R(I) = (1 - \exp(\ln(0.5) * (I / 800) ^\wedge 0.5)). \qquad (6)$$

2.2.1.2 LUMINANCE-ADAPTED RECEPTOR INSTANTANEOUS RESPONSE

The steady-state receptor response transfer function in equation (5) is non-linear over the entire range of E(I), and does not describe the observed linear response of cones to changes in luminance below the adaptation level. It appears that this equation represents the inhibition due to neural coupling among neighboring receptors, and the inherent saturation limits and non-linearities of individual luminance-adapted receptor response. The importance of this distinction is to emphasize that equation (5) does not represent the instantaneous response of the luminance-adapted cone, but represents the steady-state response of cones in their retinal context including the response of the individual receptors to input from neighbors via receptor coupling.

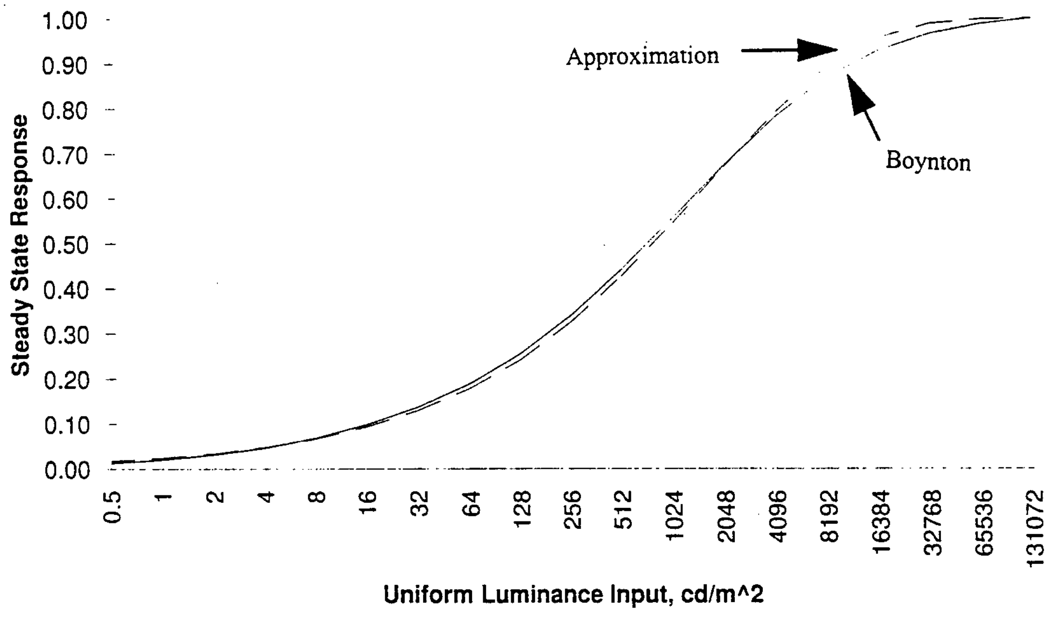


Figure 2-7. Steady state cone response as a function of uniform luminance input (derived function and numerical approximation).

Electrical receptor coupling is not a particularly well-understood aspect of retinal neurophysiology (DeValois and DeValois 1990, p. 79-80). Receptors are coupled over an area, so coupling input to an individual individual receptor is the spatial aggregate sampled over some surrounding region, i.e., the input to a cone from receptor coupling is a spatial average of the surrounding cone outputs. The effects of coupling include inhibition and summation. The degree of summation decreases with increasing illumination. Summation is one process which would contribute to the decrease in contrast sensitivity at high spatial frequencies at low luminances. Data on spatial contrast sensitivity at reduced luminance (DeValois, et al., 1974) suggest that this effect is negligible at luminance adaptation levels above 16 candela per square meter and is significant at levels below 1.6 cd/m2.

Boynton reports linear response to variations in luminance near the adaptation level and at low intensities. This is not suprising. Theoretical considerations in receptor design suggest that instantaneous receptor response should be close to linear variations in input at a level well below the input level needed to produce saturation in the receptor. Assuming for the moment that the only effect of retinal adaptation via neural coupling is inhibition, a simple form for the instantaneous receptor response, $\underline{R}$, as a function of the instantaneous luminance incident on the receptor, $\underline{I}$, is the same as that used in equation (3):

$$\underline{R}(\underline{I}, I) = \underline{E}(\underline{I}, I) \,/\, (\, E(\underline{I}, I) + X(I)\,), \qquad (7)$$

where the half-constant X(I) is a function of the luminance adaptation level, I. This functional form has the properties that it is nearly linear for low inputs, that it is nearly linear to variations in input at level near the half-constant, that it saturates at high input, and is specified by a single parameter (the half-constant).

The instantaneous energy received, $\underline{E}(\underline{I}, I)$, is equal to the instantaneous input, $\underline{I}$, multiplied by the gain resulting from pupil reflex and bleaching:

$$\underline{E}(\underline{I}, I) = \underline{I} * F(I) * S(I) \,/\, I. \qquad (8)$$

The value of the half-constant, X(I), depends on the luminance adaptation. The value of the half-constant can be derived, given the average input luminance from the steady state analysis, I, since the instantaneous output, $\underline{R}$, to the average luminance input must equal the steady-state output, R, to the average luminance input, i.e.,

$$X(I) = 631 * E(I)\text{^}0.3. \qquad (9)$$

Combining equations (7), (8) and (9) yields

$$R(\underline{I}, I) = \underline{I} / (\underline{I} + 631 * I / (F(I) * S(I))^{0.7})$$

$$= \underline{I} / (\underline{I} + X1(I)), \text{ where} \qquad (10)$$

$$X1(I) = 631 * I / (F(I) * S(I))^{0.7}. \qquad (11)$$

Figure 2-8 illustrates the transfer function represented by equation (7) by graphing instantaneous receptor response, $\underline{R}(\underline{I}, I)$, as a function of the instantaneous luminance input, $\underline{I}$, at selected adaptation levels, I. Note that the x-axis is on a log scale.

This is not quite the end of the story, because at low luminance levels, neural receptor coupling also produces summation, i.e., the response from a receptor is due in part to input from nearby receptors. This mechanism contributes to the explanation as to when contrast sensitivities at the higher spatial and temporal frequencies are attenuated more than at lower frequencies when the stimulus luminance is reduced. Data from DeValois, et al., (1974) on spatial contrast sensitivity and from Kelly (1961) on temporal contrast sensitivity suggest that this effect becomes noticeable at luminances in the neighborhood of 1.6 to 16 cd/m^2. The effect is negligible at higher luminances.

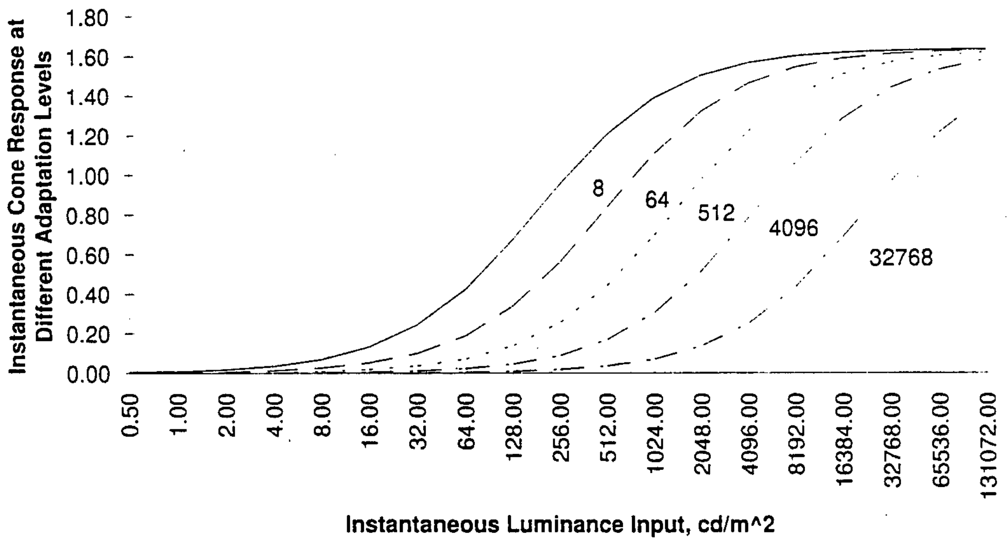


Figure 2-8. Instantaneous cone response as a function of luminance input at five different adaptation levels, assuming no summation due to receptor coupling.

Assuming that the fraction of receptor output not due to coupling can be approximated with the form of equation (3) and characterized by its half-constant, then equation (10) can be modified to account for the fraction of receptor output due to coupling summation

$\underline{R}(\underline{I}, I) = C(I) * R(I) + (1-C(I)) * \underline{I} / (\underline{I} + X1(I))$, where (12)

$C(I) = 1 - R(I) / (R(I) + R(c))$, where c is a constant in the range between 1.6 and 16 cd/m^2. (13)

Figure 2-9 illustrates the transfer function represented by equation (12) by graphing instantaneous receptor response, $\underline{R}(\underline{I}, I)$, as a function of the instantaneous luminance input, $\underline{I}$, at selected adaptation levels, I, assuming C equals 16 cd/m^2. Note that the x-axis is on a log scale.

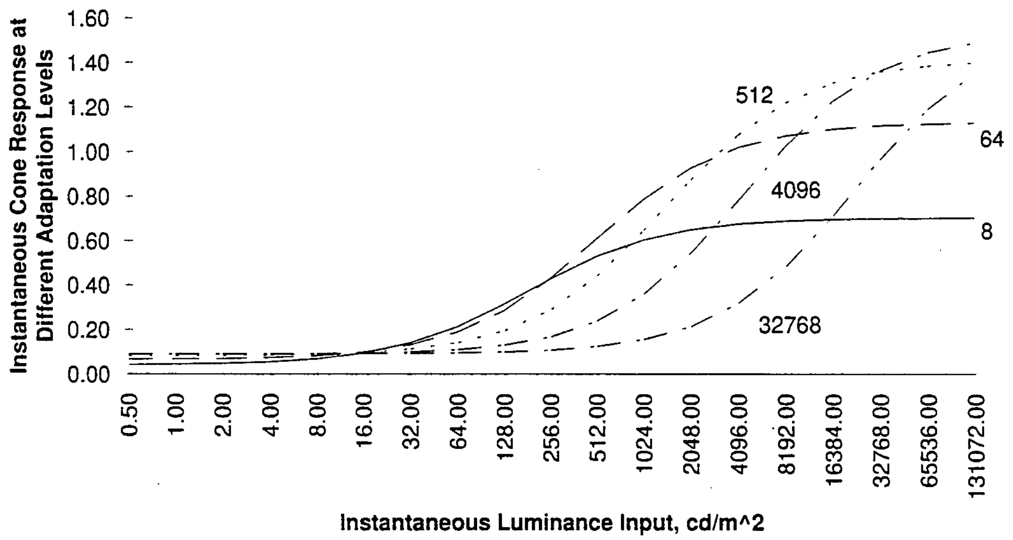


Figure 2-9. Instantaneous cone response as a function of luminance input at five different adaptation levels, with half-constant for summation due to receptor coupling at 16 cd/m^2.

2.2.1.3 APPLICATION TO COMPUTATION VISION MODELING

The current VPM early visual processing model performs a gamma correction on the input image to correct for distortions in the electronic image processing, but after that performs linear color coordinate system transformation and spatial filtering. Within the dynamic range of luminance in images used to measure contrast sensitivity, the purely linear model is adequate. However, natural scenes often contain a large luminance dynamic range. The visual system adapts to the scene luminance through pupil reflex, photopigment bleaching, and receptor coupling. Even given the adaptation, cone response is non-linear. The non-linearities are summarized in a simple equation, and should be represented in the early visual processing model.

At the present time, the NAC VPM uses a linear transformation to compute the spatio-temporal pattern of SML cone response from the spatio-temporal pattern input in the xYz CIE tri-stimulus coordinate system. In the xYz coordinate system, Y represents the luminance, while x and z specify the coordinates in color space. To represent the effects of luminance adaptation in the photopic regime, and non-linearities in receptor response, the transformation in equation (12) should be applied to the Y value (luminance) prior to the linear transformation to SML cone response.

2.2.2 SHAPE FILTERING

The shape element filtering in VPM can be generalized in a straightforward manner. The first step is to remove the current hard-wired two-orientation filtering. The second step is to implement shape filtering with a set of convolution kernels representing the different shape elements and orientations. The VPM model does not have to specify which shape elements are used; that can be a function of the image processing and analysis question. For some visual tasks, unoriented, unshaped filtering may be adequate. For other tasks, complex shapes may be appropriate. This approach facilitates research into the contribution of different shape elements by making them an input. Additionally, since most shape processing occurs on the luminance channel and very little if any occurs on the chromatic channels, the VPM can be made to mimic this allocation of processing.

The NAC VPM software was modified for the purpose of evaluating the effort required to modify the code to remove the "hard-wired" 2-orientation single-shape filtering currently in the early vision model. These changes were made for test and evaluation purposes only, and are not included in the software delivery. The following modifications were required:

1. "omi4indexMaker.map" was modified to add only the color index and a singleton placeholder; it previously added indices for color and orientation.

2. "omiCDP2OrientationBandpassInverse.map" was modified to be simply a pass through to "omiCDPBandpassInverse.map"; it previously separated the pyramid by orientation, passed both through "omiCDPBandpassInverse.map", then combined the results.

3. "omiEarlyVisionAnalyzerInverse.map" was modified to strip out a singleton index being used as a placeholder for the orientation index.

4. "omiIPDPToCDP2OrientationBandpass.map" was modified to be simply a pass-through without modifying the data.

5. "omiIPDPToCDPMultiResolution2OrientationBandpass.map" was modified to add a singleton index placeholder; previously it added orientation index and performed orientation filtering.

The "Filter" script in the current NAC VPM distribution, which runs "omiCDPFilter.map", performs convolution with an arbitrary kernel to a bandpass pyramid. This function can be used to implement spatial filtering for arbitrary shape elements. Convolution kernels for the desired shape elements will need to be specified (in the Phase II project). Also the high-level application called in the "Masks" script, omiMakeVPMPyramids.map", needs to be re-organized as a sequence of smaller applications which can be invoked via scripts from the command line. This will clarify and simplify the process of re-directing the processing flow for different signature metrics, and will make it easy to use the "Filter" command for shape element filtering.

2.3 "LATE" VISION PROCESSING MULTIPLE-HYPOTHESIS MODEL FOR SIGNATURE METRICS

The research produced a set of algorithms, computer software tools and simulation modules to evaluate a family of signature metrics. In general, different metrics will be appropriate for different visual tasks and conditions. All of the metrics are simply different ways of manipulating the output of the early vision analysis. The different manipulations result in measurement of significantly different aspects of the signature. The different metrics naturally will have different values of the calibration parameters.

The different metrics are produced by redirecting the data flow via file copy statements, and by changing software switches in input data files. The analyst can choose to compute metrics from the energy envelope (the square) of the output of the early vision analysis, or the envelope of the difference of the output of the early vision analysis for the original image and the image with the target's local contrast suppressed, and/or the difference in the texture between the target and its local surround. The analyst can choose to calculate the metrics by aggregating over the entire target region, or just the perimeter, or use one metric around the perimeter and another on the interior. The analyst can choose to normalize the signal at individual multi-resolution pixels to the eye noise power, or the combined eye noise and background modulation power. The analyst can choose to calculate the metrics based on the normalized power, or can apply a signal detection probability transformation on the individual multi-resolution pixels prior to computing the aggregate metrics. The analyst has several options to employ adaptive weights to the different visual channels in computing the aggregate metrics in order to represent the effects of prior training or knowledge regarding how the visual signature of the target differs from the background. Lastly, the analyst can model the target as a single region defined by a single perimeter, or as a network of regions, e.g., gun tube, turret and chassis, each with its own characteristic perimeter.

The processing flow and options are illustrated in the flow charts and equations presented in Figures 2-10 through 2-23. Figure 2-10 shows the high-level flow for processing dynamic (moving target) data from a sequence of digitized video images. Figure 2-11 shows the high-level flow for static or temporally-filtered image processing. Figures 2-12 through 2-23 show the next-level flowcharts for each of the blocks in Figure 2-11.

Signature metrics for automotive conspicuity results will not necessarily be appropriate for other visual tasks and conditions such as military target acquisition. A robust and economical perception testing approach based on the method-of-adjustments is proposed for metric screening and preliminary calibration in Phase II. The proposed method involves two complementary screening criteria: correspondence ranking the gross level of signature level for task performance (e.g., high, medium and low), and one to order the differences between pairs of images within the same gross rank. In the same exercise, the subjects can be asked to respond for several different tasks, so that a single test can produce data related to search, detection and recognition processes.

For any given visual task and context, there may be a single dominant metric or a combination of several metrics may be required. For detecting highly conspicuous cars, many of the metrics may prove to be highly correlated, and a single metric may have sufficient explanatory power. For detecting low-signature military targets in varied and unusual field situations, several different metrics may be needed to represent the different bases for detection.

In the case of a "simple" target, it may be appropriate to apply the metric to the entire target, as in the case of detecting an oncoming automobile. In cases of a "complex" target, perception of a part is sufficient and the whole target need not be perceived in its entirety, e.g., a tank section in which detection of one tank is sufficient to detect the section, or for some tasks and situations it may be appropriate to consider the tank to be a collection of parts (e.g., a gun tube, a turret and a chassis). Detecting any one of the parts is sufficient for detecting the vehicle, and detecting some combination of the parts is sufficient to recognize the vehicle. In these cases it may be more appropriate to analyze the target as a collection of components, rather than as a single coherent object.

Further research in Phase II is needed to screen and evaluate the different metrics and analysis processing to determine their applicability to different tasks.

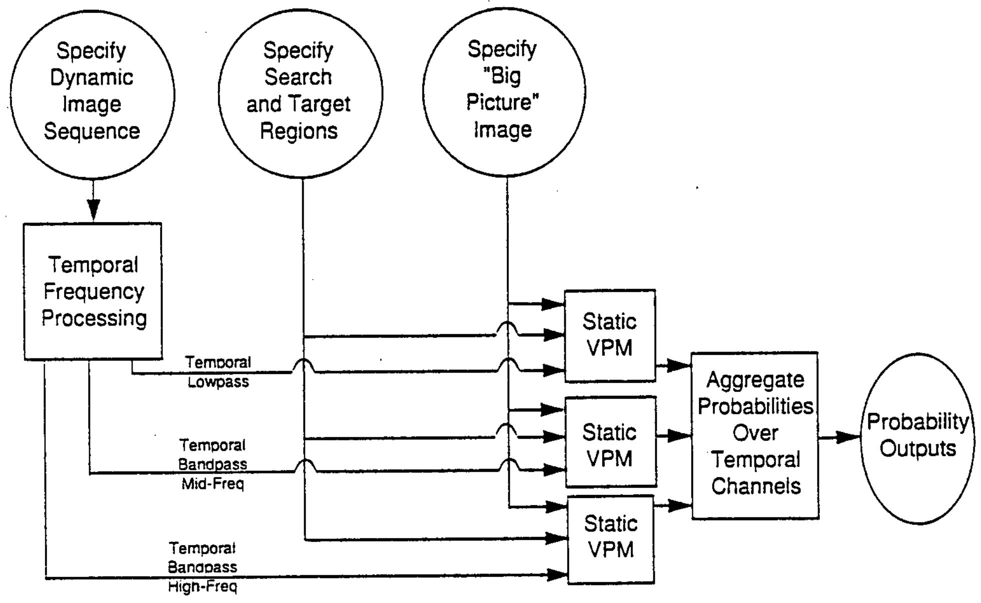

Specify Dynamic Image Sequence
Specify Search and Target Regions
Specify "Big Picture" Image
Temporal Frequency Processing
Temporal Lowpass
Temporal Bandpass Mid-Freq
Temporal Bandpass High-Freq
Static VPM
Static VPM
Static VPM
Aggregate Probabilities Over Temporal Channels
Probability Outputs

Figure 2-10. NAC VPM dynamic processing flow.

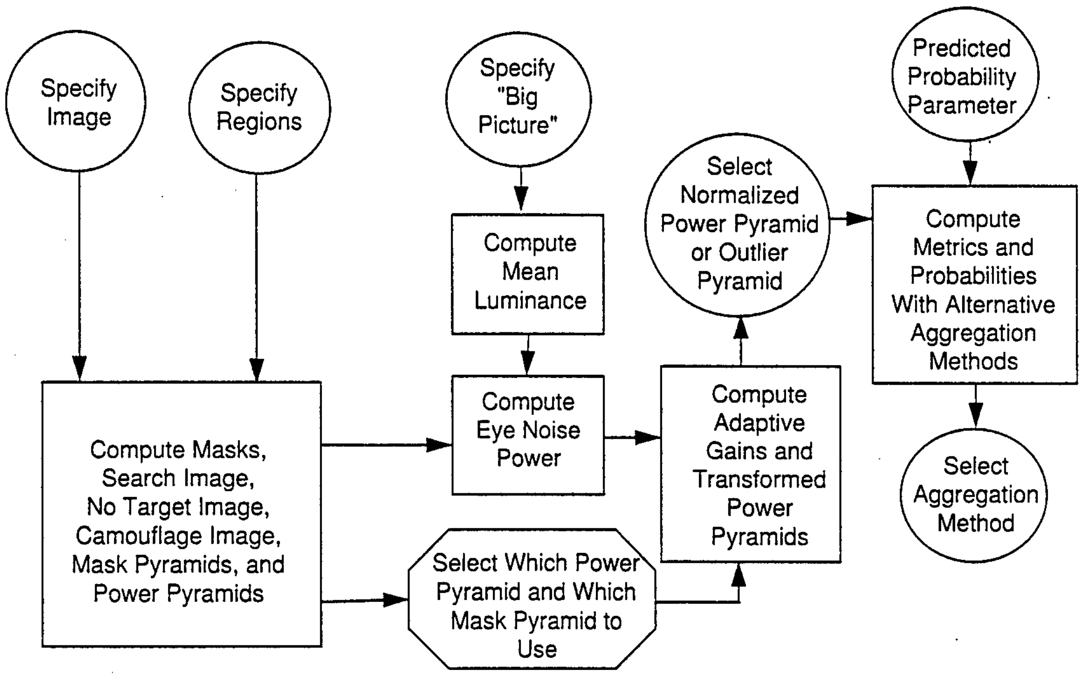

Specify Image
Specify Regions
Specify "Big Picture"
Predicted Probability Parameter
Select Normalized Power Pyramid or Outlier Pyramid
Compute Metrics and Probabilities With Alternative Aggregation Methods
Compute Mean Luminance
Compute Eye Noise Power
Compute Adaptive Gains and Transformed Power Pyramids
Select Aggregation Method
Compute Masks, Search Image, No Target Image, Camouflage Image, Mask Pyramids, and Power Pyramids
Select Which Power Pyramid and Which Mask Pyramid to Use

Figure 2-11. NAC VPM static processing flow.

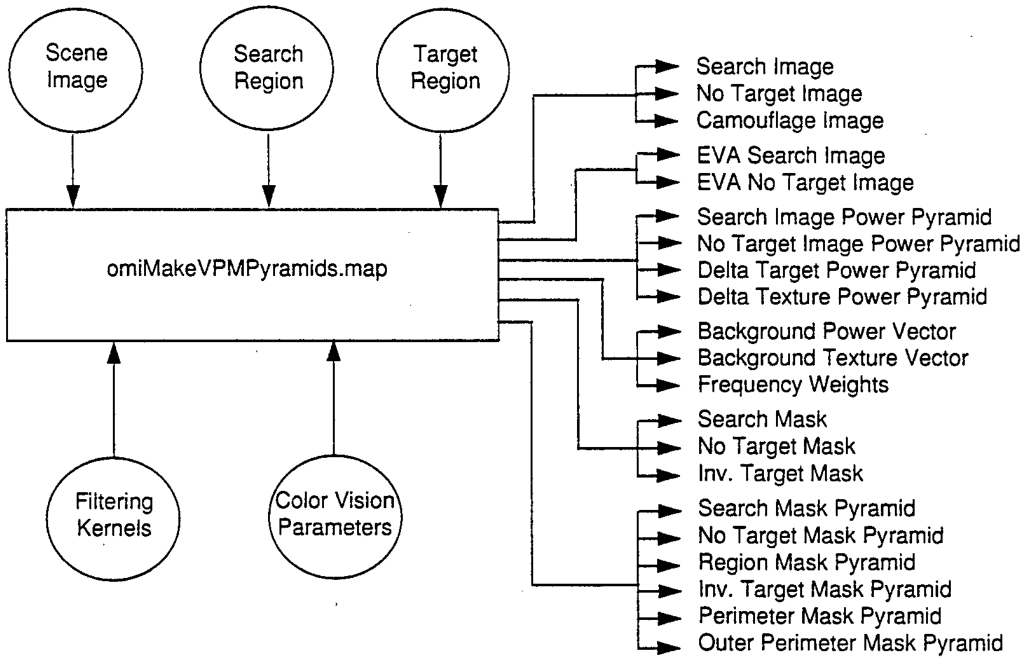


Figure 2-12. Generate masks and pyramids.

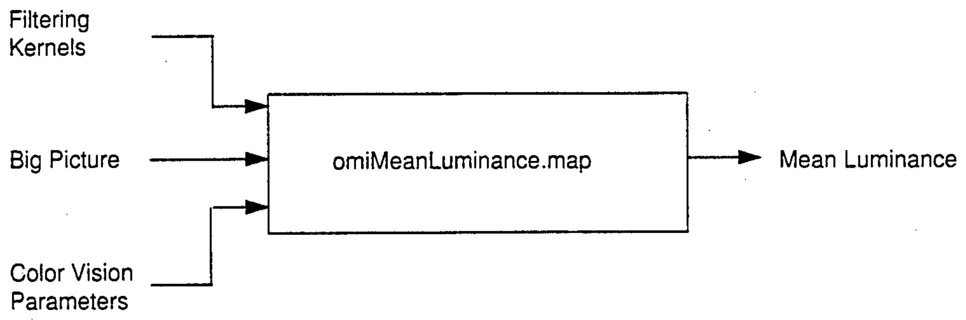


Figure 2-13. Compute mean luminance.

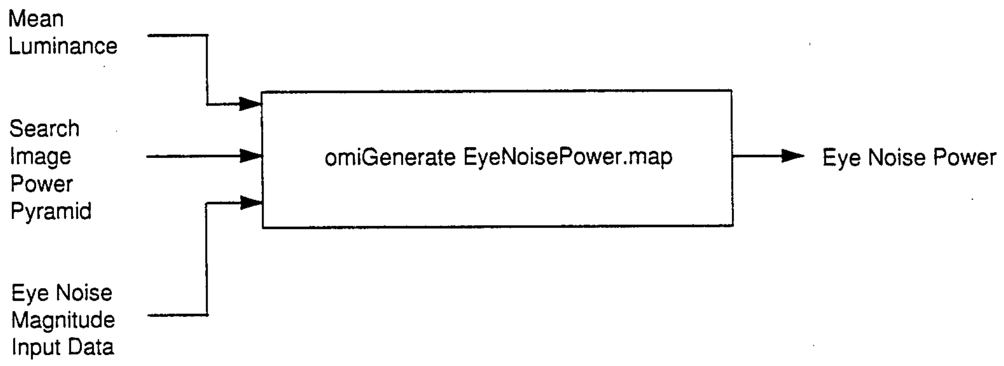

Mean Luminance
Search Image Power Pyramid
Eye Noise Magnitude Input Data
omiGenerate EyeNoisePower.map
Eye Noise Power

Figure 2-14. Compute eye noise power vector.

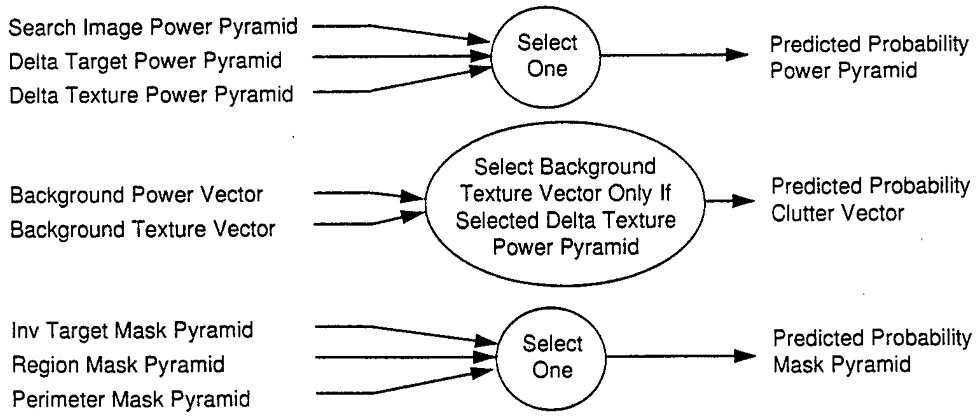

Search Image Power Pyramid
Delta Target Power Pyramid
Delta Texture Power Pyramid
Select One
Predicted Probability Power Pyramid
Background Power Vector
Background Texture Vector
Select Background Texture Vector Only If Selected Delta Texture Power Pyramid
Predicted Probability Clutter Vector
Inv Target Mask Pyramid
Region Mask Pyramid
Perimeter Mask Pyramid
Select One
Predicted Probability Mask Pyramid

Figure 2-15. Select pyramid and mask pyramid.

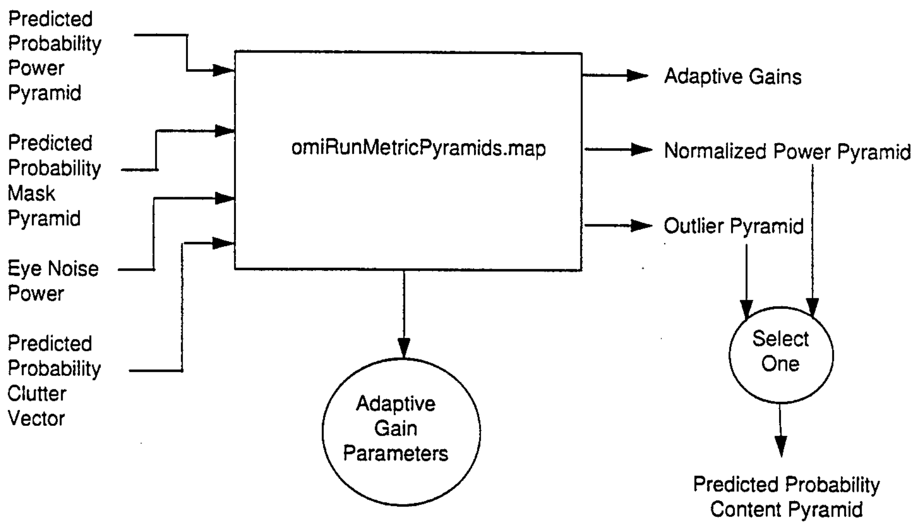

Predicted Probability Power Pyramid
Predicted Probability Mask Pyramid
Eye Noise Power
Predicted Probability Clutter Vector
omiRunMetricPyramids.map
Adaptive Gains
Normalized Power Pyramid
Outlier Pyramid
Select One
Adaptive Gain Parameters
Predicted Probability Content Pyramid

Figure 2-16. Compute adaptive gains and transformed power pyramids.

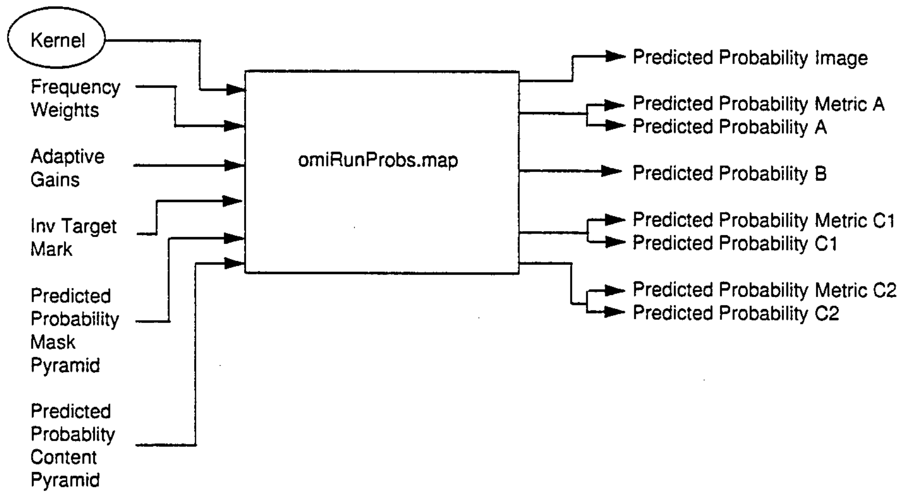

Kernel
Frequency Weights
Adaptive Gains
Inv Target Mark
Predicted Probability Mask Pyramid
Predicted Probablity Content Pyramid
omiRunProbs.map
Predicted Probability Image
Predicted Probability Metric A
Predicted Probability A
Predicted Probability B
Predicted Probability Metric C1
Predicted Probability C1
Predicted Probability Metric C2
Predicted Probability C2

Figure 2-17. Compute metrics and probabilities with alternative aggregation methods.

- Parameters α and β of the cumulative distribution function used to compute the outlier pryamid

$$P_{i,j,c} = 1 - e^{\ell n(1/2) \cdot (X_{i,j,c}/\alpha)^{\beta}}$$

where $P_{i,j,c}$ is the probability the normalized power ($X_{i,j,c}$) at RF position (ij) on visual channel C is an outlier

- 0/1 switch parameters W,X,Y, and Z of the adaptive gain equation

$$\text{Adaptive Gain}_c = \frac{(\overline{X}_c - W \cdot \text{BkgPower})^{y}}{(\text{EyeNoise} + X \cdot \text{BkgPower})^{z}}$$

where $\overline{X}_c$ is the mean energy on visual channel C, averaged over the Predicted Probability Mask Pyramid region

- 0/1 switch parameter, δ, regarding using Predicted Probability Clutter when normalizing Predicted Probability Power Pyramid

$$N_{i,j,c} = X_{i,j,c} / (\text{EyeNoise} + \delta \cdot \text{Pr ed Pr obClutter})$$

where $N_{i,j,c}$ is the normalized power at RF location (i,j,c) on visual channel C, and $X_{i,j,c}$ is the unnormalized power

Figure 2-18. Parameters in the "Adaptive Gain Parameters" Input file, and their use in VPM processing.

$$\mathrm{PredProbImage} = \left(\sum_{\substack{\text{visual}\\ \text{channel}\\ c}} \mathrm{AdaptiveGain}_C \cdot \mathrm{FreqWeight}_C \cdot \mathrm{Expand}(\mathrm{PredProbContentPyramid}_C)^T \right)^{1/T}$$

- Predicted Probability Image is an image of the Tth root of the weighted sum of the Tth power of the Predicted Probability Content. Predicted Probability Content Pyramid planes are expanded to full size, and weighted by the product of the adaptive gain for the channel and the frequency weight. The frequency weight just scales values down to compensate for expansion to maintain a constant weight per receptive field
- Predicted Probability Image is a spatial map of the cumulative content over all visual channels

Figure 2-19. Predicted probability image computation.

- $\mathrm{Metric\ A} = \sum_{\substack{\text{inverse}\\ \text{target}\\ \text{mask}}} \left(1 - e^{\ell n(1/2)\cdot(\mathrm{PredProbImage}/\alpha)^{\beta}} \right)$

  Metric A is the expected number of detectable pixels on the target computed from the cumulative content in the Predicted Probability Image.
- $\mathrm{PredProb\ A} = 1 - e^{\ell n(1/2)\cdot(\mathrm{Metric\ A}/\alpha)^{\beta}}$

  Predicted Probability A is an empirical prediction of the probability of detection based on the expected number of detectable pixels.

Figure 2-20. Metric and probability A computation.

- Metric B calculation first integrates Predicted Probability Content Pyramid on individual channels over the Predicted Probability Mask Pyramid, then estimates a probability of detection on each channel, then pools the Pds over the channels

$$\text{VisualChannelSumStats}_C = \sum_{\substack{i,j \\ \text{multi-} \\ \text{resolution} \\ \text{plane}_C}} X_{i,j,c} \cdot Y_{i,j,c}$$

where $X_{i,j,c}$ is the Predicted Probability Content at location (i,j) on channel C, and $Y_{i,j,c}$ is the Predicted Probability Mask value at location (i,j) on channel C

$$\text{Prob B on Channel}_C = 1 - e^{\ln(1/2)(\text{VisualChannelSumStats}_C/\alpha)^{\beta}}$$

$$\text{PredProbB} = 1 - e^{\left(\sum_C \text{AdaptiveGain}_C \cdot (1-P_C)^T\right)^{1/T}}$$

Figure 2-21. Metric B.

$$\text{MetricC1} = \sum_{\substack{\text{inverse} \\ \text{target} \\ \text{mask}}} \text{PredProbImage}$$

$$\text{PredProbC1} = 1 - e^{\ln(1/2)(\text{MetricC1}/\alpha)^{\beta}}$$

Metric C1 integrates Predicted Probability Image over the target region. This metric is input to an empirical cumulative distribution function to predict the probability

Figure 2-22. Metric and probability C1.

$$\text{Metric C2} = \left(\sum_{\substack{\text{visual}\\ \text{channel}\\ C}} \left(\text{AdaptiveGain}_C \cdot \sum_{\substack{i,j\\ \text{onmulti-}\\ \text{resolution}\\ \text{plane}_C}} X_{i,j,c} \cdot Y_{i,j,c} \right)^T \right)^{1/T}$$

where $X_{i,j,c}$ is the Predicted Probability Content at RF location (i,j) on visual channel C, and $Y_{i,j,c}$ is the Predicted Probability Mask Pyramid

$$\text{PredProbC2} = 1 - e^{ln(1/2)(\text{MetricC2}/\alpha)^{\beta}}$$

Metric C2 is the nth root of the sum of the nth power of the visual channel sum statistics, weighted by the adaptive gain. The probability C2 is computed with a generic psychometric function. α is the mean, and β characterizes the width

Figure 2-23. Metric and probability C2.

3.0 SOFTWARE USER'S MANUAL

3.1 INSTALLATION

3.1.1 RETRIEVAL FROM TAPE

Log in as "root" to ensure that you have permissions to create directories and executables.

Create a target directory into which to load the files and change to that directory, or simply change to the target directory if one already exists:

```
mkdir <dir-name>

cd <dir-name>
```

Find the <unit #> of the tape device in the response to the instruction "hinv". The <unit #> is normally a single digit number. Untar the tape with the following instruction:

```
tar -xvf /dev/rmt/tps1d<unit #>
```

Confirm that the following environmental variable settings are active:

```
setenv OMISRC_HOME <dir-name>/src

setenv PATH ${PATH}:${OMISRC_HOME}/nac-vpm/bin
```

Issue the following instructions to complete the installation:

```
cd src/nac-vpm

rehash

VPMinstall
```

3.1.2 ADDING OR UPDATING A USER

Ensure that the following environmental variable settings are active:

```
setenv OMISRC_HOME <dir-name>/src

setenv PATH ${PATH}:${OMISRC_HOME}/nac-vpm/bin
```

where <dir-name> is the directory in which NAC VPM src was installed.

To run NAC VPM in a directory, e.g., <working-dir>, the directory must be set up to run NAC VPM:

```
cd <working-dir>

VPMsetup.
```

These instructions will copy start-up data files and make symbolic links in the target directory. It is possible that under some operating system configurations, the user may be asked whether or not to permit making each of the links. If this occurs, respond "y" for yes.

3.2 OPERATION

NAC VPM is intended to be both a visual performance model and a workbench for research on and development of visual processing simulations and visual performance models. Consequently, there are simultaneous needs for "turn-key" operation and for "on-line" modification and refinement. In order to meet these diverse needs, NAC VPM has been designed for the following three levels of interaction:

1. Turn-key operation by executing UNIX command-line instructions, including both standard UNIX instructions and executable files containing canned NAC VPM shell scripts.
2. Interface configuration and high-level data flow control by editing the canned NAC VPM shell scripts or entering sequences of UNIX file-manipulation instructions to execute flow-control.
3. Algorithm refinement and low-level data flow modification by editing the map files which define the data flow to the execution engine.

The highest level of interaction requires little knowledge beyond the meaning of the commands. The middle level requires understanding of the high-level data flows and the intermediate files produced by the executable modules. The lowest level requires some understanding of the model data flows and the underlying software development workbench.

3.2.1 TURN-KEY OPERATION

3.2.1.1 SELECT AND CONVERT AN IMAGE FOR ANALYSIS.

If the image is already in the NAC VPM format (e.g., produced by the temporal preprocessor or other NAC VPM programs), then simply copy it to the "SceneImage" file in the current working directory with the instruction

```
cp <path:filename> SceneImage
```

If the image is not in the NAC VPM ALPHA format, it will need to be converted. If the image is in native SGI format ("*.rgb"), then convert it to the "SceneImage" file with the instruction

```
Convert <path:filename>
```

If the image is in some other format, then display the image and record it from the screen with the "snapshot" SGI utility, or use one of the many image file format conversion utilities provided by SGI to convert it to the "*.rgb" format (e.g. arf2sgi). This will produce a "*.rgb" file which can be converted with the "Convert <path:filename>" instruction.

Before converting the image, it should be cropped to exclude the portion of the image outside the area of interest, i.e., outside the region where an observer is presumed to be looking for the target. This will reduce the processing time. In general, the height and width of the cropped image should be three to five times the target height and width.

3.2.1.2. SPECIFY THE SEARCH REGION.

Designate the search region, i.e., the portion of the scene in which an observer would look for the target, by entering the instruction

```
EdSearch
```

This brings up a menu to view the image and edit the search region. Select either the "Color" or "Scaled Color" option with the mouse. Select "Remove" to remove the previous search region. Select "Add" to add a new search region. Denote the vertices (corners) of the search region on the image by positioning and click the mouse. "Undo" removes the last entry. The search region is specified as an irregular polygon, or as the union or intersection of irregular polyhedra. Polygons are automatically closed. The search region is saved in the "SearchRegion" file. The search region is not saved until the search region editor menus are closed. Pressing "Ctrl-C" will abort the process and the changes will not be saved.

In general, it is helpful to use the SGI utility program "snoop" in conjunction with EdTarget and EdSearch. Snoop dynamically updates a magnified image of the region centered at the cursor location on the screen. Simply issue the instruction "snoop" in an active window.

3.2.1.3 SPECIFY THE TARGET REGION

The target region contains the target of interest. It should contain all of the target. If pixels overlap both the target and the surround, they should be included inside the target region. In general, the target region is inside the search region. Any portion of the target region which is outside of the search region is disregarded. The instruction

EdTarget

brings up a menu to view the image and edit the target region. The operation of the utility is the same as when it is used for editing the search region. The target region is saved in the "TargetRegion" file.

3.2.1.4 EVALUATE A SIMPLE METRIC AND ESTIMATE THE PROBABILITY OF DETECTION

Two instructions are needed to evaluate one specific metric and compute the probability of detection using calibration data based on the calibration data in the input files. Issue the following instructions to execute the command scripts:

Masks

Metrics

The instruction "Masks" computes all of the images and image pyramids previously described in Section 2. The instruction "Metrics" contains a sequence of file copy commands and VPM script commands to execute a particular data flow, as described in Section 2. The particular data flow choices can be altered by editing the "Metrics" script file. The probability and metric outputs are put in the file "PredProbOutput.dat."

The instructions access several input data files:

EVMInputData

EyeNoiseMagnitudeInputData

AdaptiveGainParameters

PredProbParameters.

The EVMInputData file contains data specifying the gamma correction parameters which relate the pixel values in the digital image to luminance as seen by observers. The EyeNoiseMagnitudeInputData file contains data specifying the instantaneous field of view (IFOV-the angle subtended by a single pixel). These data can and should be edited, as appropriate for the digital images being analyzed. Modifying any other data in these files or in the AdaptiveGainParameters will invalidate the calibration parameter settings. PredProbParameters contains the calibration data used to compute probabilities from signature metrics.

3.2.2 INTERFACE CONFIGURATION AND HIGH-LEVEL DATA FLOW CONTROL

The first three instructions, to convert a file and to edit the search and target regions, are performed as in the turn-key operation. The following options replace the turn-key "Metrics" instruction.

3.2.2.1 GENERATE MASKS AND REFERENCE IMAGES

The instruction following instruction is issued to create a number of mask and reference images used to compute a variety of different metrics. This instruction is time-consuming because it computes a number of different intermediate files which are used to branch out to examine different metric options. In future upgrades, this all-purpose instruction should probably be supplemented with instructions which will generate specific combinations of masks and reference images for different specific metrics.

Masks

This operation reads data in input text files, which can be viewed and edited with any text editor. The EdgeDetectionParameters file holds parameters used in creating the edge mask for computing edge metrics. In general, these data will not be altered after model calibration. EVMInputData holds the photometry parameters (gamma correction parameters and RGB to CIE xyz coordinate system parameters), and the parameters to convert the CIE xyz coordinate system to the eye's luminance/color-opponent coordinate system. In general, the photometry parameters will need to be adjusted for different image data.

The Masks instruction produces a large number of output files:

The SearchImage is the SceneImage with the visual information outside of the SearchRegion removed. The content of the SearchRegion is blended-out using the non-stationary multi-resolution filtering and reconstruction algorithm. The EVASearchImage is the multi-resolution output of the early vision analysis (EVA) of the SearchImage. The SearchImagePowerPyramid is the square of the EVASearchImage.

The NoTargetImage is the SceneImage with the visual information inside the TargetRegion and the outside of the SearchRegion removed. The content of the SearchRegion is blended-in to the TargetRegion. The EVANoTargetImage is the multi-resolution output of the early vision analysis (EVA) of the NoTargetImage. The NoTargetPowerPyramid is the square of the EVANoTargetImage.

The TargetMask is a single plane image equal to one outside the TargetRegion, equal to zero inside the TargetRegion, and equal to zero on the TargetRegion boundary.

The SearchMask is a single plane image equal to one inside the SearchRegion, equal to zero outside the SearchRegion, and equal to one on the boundary. SearchMaskPyramid is the output of multi-resolution spatial filtering applied to the SearchMask.

The NoTargetMask is the product of the TargetMask and the SearchMask. The NoTargetMaskPyramid is the output of multi-resolution spatial filtering applied to the NoTargetMask.

The InvTargetMask is equal to one minus the TargetMask. The InvTargetMaskPyramid is the output of multi-resolution spatial filtering applied to the InvTargetMask.

The EdgeMaskPyramid is a multi-resolution stack of images of the probability of edge.

The CamouflagedImage is identical to the SearchImage outside of the TargetRegion, but has suppressed the signature and added background-like texture to the interior of the target region. The CamouflagedImage is a final output, and is not used in any further computation.

The RegionMaskPyramid is a multi-resolution stack of images designating the strict interior of the TargetRegion at each multi-resolution level of spatial filtering.

The PerimeterMaskPyramid is a multi-resolution stack of images designating the perimeter of the TargetRegion at each multi-resolution level of spatial filtering.

The DeltaTargetPowerPyramid is the square of the difference between the EVASearchImagePyramid and the EVANoTargetPyramid. The BkgPowerVector is the root-mean-square (RMS) of the NoTargetPowerPyramid evaluated on the PerimeterMaskPyramid.

The DeltaTexturePowerPyramid is the absolute value of the SearchImagePowerPyramid minus the BkgPowerVector. The BkgTextureVector is the root-mean-square (RMS) of the DeltaTexturePyramid evaluated on the PerimeterMaskPyramid.

All of the images and pyramids can be viewed using the instruction

```
View <file1> ... <fileN>
```

The multi-resolution pyramid images can also be automatically expanded to full size before display with the instruction:

```
EView <filename>
```

The BkgPowerVector and the BkgTextureVector can be viewed with any text editor.

3.2.2.2 COMPUTE MEANLUMINANCE AND EYENOISEPOWER

The analysis of visual information is, in part, relative to the internal noise of the visual system on the different visual channels. The EyeNoisePower is influenced by the mean luminance of the scene due to the adaptive mechanisms of the early visual system. The mean luminance is computed from the image in the file named BigPicture. This image should, ideally, cover a large visual angle, e.g., 90 degrees by 90 degrees. It can have low spatial resolution since it is only used to compute the average luminance. If the luminance of the scene to which the eye is adapted does not differ significantly from the luminance in the SceneImage, then it is sufficient to copy SceneImage to BigPicture. If the SceneImage were cropped from a larger image, then the larger image could be converted and copied to BigPicture.

The EyeNoisePower also accesses input data in the EyeNoiseMagnitude-InputData. The EyeNoiseMagnitudeInputData contains data on the viewing situation and visual contrast sensitivity. Three parameters may need to be changed for the analysis. The first parameter, T, should be set to t0 for analyzing static or temporal lowpass images, or to t1 or t2 for temporal mid-frequency and high-frequency, respectively. The fifth parameter, IFOV, is the angular subtense of a pixel in degrees. The seventh and last parameter, A, is the number of degrees off of the visual axis at which the scene is viewed (nominally zero). The visual contrast sensitivity parameter values will not, in general, be changed. The baseline contrast sensitivity parameter values are for photopic luminance adaptation. It is possible to provide additional data values for mesopic and scotopic luminance adaptation, although these files have not been prepared at this time.

The following instructions cause the system to compute EyeNoisePower and MeanLuminance output files:

```
cp SceneImage BigPicture

EyeNoise
```

3.2.2.3 SELECT MASK AND PYRAMID FILES TO USE TO COMPUTE METRICS

One of the ways a researcher can explore different metrics is to select different outputs from Masks, to use to compute the metrics.

One of the following three files needs to be copied to the PredProbPowerPyramid file:

SearchImagePowerPyramid

DeltaTargetPowerPyramid

DeltaTexturePowerPyramid

If DeltaTexturePowerPyramid is selected, then BkgTextureVector should be copied to PredProbBkgVector, else BkgPowerVector should be copied.

One of the following mask pyramids needs to be copied to PredProb-MaskPyramid:

InvTargetMaskPyramid

PerimeterMaskPyramid

RegionMaskPyramid

One of the following masks needs to be copied to PredProbMask:

InvTargetMask

PerimeterMask

3.2.2.4 SET ADAPTATION PARAMETERS (OPTIONAL)

NAC VPM represents the effects of adaptation on the visual channels, as specified by the values in the data in the AdaptiveGainParameters file. It can be edited with any text editor. One of the ways that a researcher can construct different metrics is to change the settings in the AdaptationParameters file.

The NormalizationBkgGain should be set to zero to normalize the visual power to the EyeNoisePower, or set to one to normalize to the combined EyeNoisePower and BkgPowerVector.

The following formula is used to compute the adaptive gains:

(TPV - X * BPV)^Y / (ENP + W * BPV)^Z,

where TPV is the mean power over the target region, BPV is the BkgPowerVector, and ENP is the EyeNoisePower. These are computed by the software. The inputs are X, the NumeratorBkgPowerGain, Y, the NumeratorExponent, W, the DenominatorBkgPowerGain, and Z, the DenominatorExponent. The default values of W, X, Y and Z are all zero.

The last two parameters in the AdaptiveGainParamters file are the mean and exponent for the cumulative distribution function to transform the normalized signal (at each multi-resolution pixel) to a probability of detecting the signal at the multi-resolution pixel from the noise process. (Note that the noise process can be either the eye noise or combined eye noise and background clutter depending on the setting of the NormalizationBkgGain parameter.)

3.2.3 ANALYZE THE PYRAMIDS

Issue the instruction "cle -f omiRunMetriPyramids.map" to analyze the pyramids. Two output files are produced:

NormalizedPowerPyramid

OutlierPyramid.

One of these should be copied to PredProbContentPryamid. They can be viewed with View or Eview.

3.2.4 COMPUTE METRICS AND PROBABILITIES

The instruction "cle -f omiRunProbs.map" computes the signature metrics and predicted probabilities. It produces the "PredProbImage" showing the spatial distribution of visual information, as an aid in understanding what the particular choice of data flow is computing. Four methods of aggregation are used, as described in Section 2. The numerical outputs are written to the following text files:

PredProbMetricA

PredProbA

PredProbB

PredProbMetricC1

PredProbC1

PredProbMetricC2

PredProbC2.

The text files can be examined with any editor or the "cat" command. The PredProbImage can be examined with the "View" instruction.

3.2.5 POST_PROCESSING UTILITIES

Several post-processing utilities are provided to examine images and to construct stimuli for perception testing.

Spotlight performs spatial analysis of variance and thresholding on images (multi-resolution spatial frequency bandpass filtering on the image, squares the output thresholds each plane at N times the standard deviation of the plane). The factor N is the same for all planes, and is read from the Spotlight input file. The output is named SpotlightImage.

Spotlight <image file name>.

Two utilities are provided to blend or mix images.

MultiResMixImages <imageA filename> <imageB filename>.

This utility constructs an output image from two input images. The output image is reconstructed from a linear combination of the multi-resolution decompositions of the two input images. The filenames are specified on the command line. The weights are specified in three files:

FreqWeights.dat

OrientWeights.dat

ColorWeights.dat

For each plane, the weights for the two images sum to unity. The weight for image A on a given plane is the product of the frequency weight, the orientation weight, and the color weight. (The input image files are copied to MRMA.img and MRMB.img).

The output image is stored in the file MultiResMix.img.

MaskMixImg <imageA filename> <imageB filename> <mask.img>

This utility blends the two input images using the mask image and a mix ratio (a number between zero and one): specified in the MixRatio.dat file.

Two output images are produced:

DeepMix.img

FlatMix.img.

DeepMix.img is made by applying the multi-resolution decomposition of the mask to the multi-resolution decompositions of the images, and then reconstructing. The FlatMix.img is made at the image level, without multi-resolution decomposition.

3.2.6 DYNAMIC SCENE ANALYSIS

Create the "temporal.sequence" file (an example is provided in the sample directory) which contains the list of the sequence of filenames containing the sequence of video frames, and marks the point in time at which to sample the filtered time sequence. Then issue the instruction:

```
TimeSequence
```

This produces three output images in NAC VPM format, named SceneTHigh.img, SceneTMid.img, and SceneTLow.img.

At the present time the dynamic analysis has not been consolidated into a single command. Dynamic analysis requires analyzing each of the three temporally-filtered images individually, then manually combining the results, as shown in Section 2. All of the following actions will be packaged as a single instruction during Phase II of the project.

Dynamic scene analysis involves analyzing each of the three output images produced by the TimeSequence command as though they were individual static images with the following considerations:

1. The SearchRegion and TargetRegion need only be created once. The same regions are to be used for all three temporally filtered images.
2. The BigPicture file (which is used to compute the mean luminance of the scene) need only be defined once. The same BigPicture is to be used for all three temporally filtered images.
3. Analyze each temporally filtered image following the same steps as for static images, then combine the results using a probability summation rule (The probability of detection is equal to the probability of detection on any one temporally filtered image. Assuming detection on different temporal bands is independent, the net probability is equal to one minus the product of one minus the probabilities of detection over the three temporally filtered images).

3.3 MODIFICATION

The software is implemented as UNIX shell scripts and data flow maps running on the Vision Simulation Workbench. The UNIX shell scripts and data flow maps are all text files which can be edited with any text editor. Existing files can be modified, or new files created with the editor. The UNIX shell scripts copy files around with copy ("cp") to change the names to the names of the temporary files used in the Workbench data flow maps. The UNIX shell scripts also invoke the command line engine ("cle") to execute top-level data flow map applications.

The Vision Simulation Workbench constructs an executable program from the data flow maps, and executes it. The data flow maps are hierarchical: they employ instances of lower level modules which can be either other data flow maps or C++ native data types. Appendix B lists the module type names, input ports and output ports for the currently available general purpose modules. Appendix C lists the module type names and I/O ports for the application-specific modules used on NAC VPM. The module names are generally descriptive of their function: what data type they take as input, what operation they perform, and what data type they produce as output. "IPDP" stands for "Image Plane Data Packet," a single plane image. "CDP" stands for "Channel Data Packet," an associative array of any other data type. "Float" stands for a floating point scalar. "Matrix" stands for a matrix of floating point scalars.

The command line to cause the workbench to construct and execute a program is

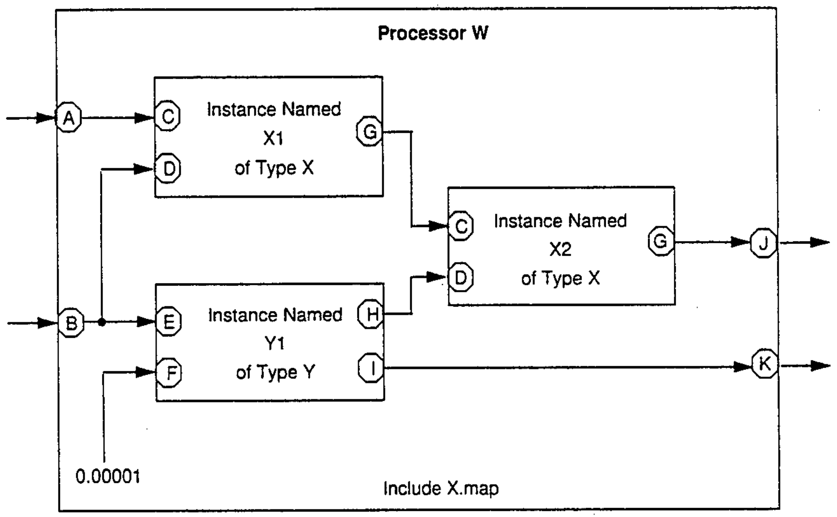
```
cle -f XYZ,
```

where XYZ is the name of a file containing the statements specifying a top-level application processor (a "Universe" processor). The command line engine back-chains from the statements which cause outputs to be written to files. Outputs can be a data type in the workbench, e.g. floats, matrices, image plane, and associative arrays of data types. At the present time, the only type of module which writes to files is the "omiASCIIFileOutput" type, so at least one instance of this type must be in the Universe to initiate processing.

Figure 3-1 illustrates the syntax of the Vision Simulation Workbench language and the correspondence between the language and a data flow diagram. Figure 3-2 provides a formal definition of the syntax. The example in the figure creates a new type of module (the name of the type is "W"). The module is defined by a data flow map stored in the text file named "W.map." Comment lines are preceded by double slashes ("//").

As a matter of convention, the name of the text file specifying the module is the name of the module (e.g., "W") with the ".map" suffix appended. The exception to this convention is in the case of the top level application processes. The top-level application process file names can be any legal UNIX file name, but the processor name must be "Universe".

Module type W has two input ports (named "A" and "B"), and two output ports (named "J" and "K"). The module type name is assigned in the "processor" statement. Module type W uses two instances of module type "X" (named "X1" and "X2") and one instance of module type "Y" (named "Y1").


Processor W
Instance Named X1 of Type X
Instance Named X2 of Type X
Instance Named Y1 of Type Y
A
B
C
D
G
E
F
H
I
C
D
G
J
K
0.00001
Include X.map

```
//text in file "W.map", include X.map
Processor W
{
//name instances of module types
instance X          X1
instance X          X2
instance Y          Y1
//define input ports
alias               X1:C    to    :A
alias               X1:D    to    :B
alias               Y1:E    to    :B
default             Y1:F    to    packet
    {begin omiFloatDataPacket 0.00001 end omiFloatDataPacket
//internal connections
connect             X1:G    to    X2:C
connect             Y1:H    to    X2:D
//define output ports
alias               X2:G    to    :J
alias               Y1:I    to    :K
}
```

Figure 3-1. Example data flow diagram and corresponding vision simulation workbench script.

```
Comment line:
        // <string>

Include ".map" files defining other processors:
        include <filename>

Define a processor type:
        processor <process type name> { ... <statments> ... }

Define an application:
        processor Universe  { ... <statments> ... }

Create a local instance of a process and give it a local name:
        instance <process type name> <local instance name>

Wire the output of one local instance to the input of another:
        connect <local instance name>:<output port name> to <local instance name>:<input port name>

Wire the input of one local instance to the output of another:
        connect <local instance name>:<input port name> to <local instance name>:<output port name>

Link the output of a local instance to an output of the process type
        alias <local instance name>:<output port name> to :<output port name for this process type>

Link the input of a local instance to an input of the process type
        alias <local instance name>:<input port name> to :<input port name for this process type>

Specify the value to use on an input port to a local instance if that port is not otherwise connected or aliased:
        default <local instance name>:<port name> to packet { <data packet> }

Set the value of an constant:
        define <local instance name>:<symbol> to <string>

Set the initial value of an internal variable
        initialize <local instance name>:<symbol> to packet { <data packet> }
```

Figure 3-2. Vision simulation workbench formal grammar.

In this example, X is a data flow map module created in the workbench language, and Y is a native type C++ module. The defining data file "X.map" needs to be specified in an "include" statement, to instruct the workbench to search for the file. Nothing needs to be included to use instances of native type C++ modules.

Instances of type X have input ports named "C" and "D", and an output port named "G". Module W uses two instances of type X, so it instantiates X twice with different names ("X1" and "X2") using the "instance" statement. The names of the input and output ports do not change, since they are attributes of the type, not the instance.

The input ports of type W are named "A" and "B". They are connected directly to the input ports of X1 and Y1 via the "alias" statements. Note that the input to X on port B is piped to both X1 and Y1 with two separate "alias" statements. Similarly, the output ports of type X ("J" and "K") are connected directly to the output ports of X2 and Y1 via "alias" statements. The "alias" statements define the names of X's input and output ports.

The outputs of X1 and Y1 are connected to the inputs to X2 via the "connect" statements. One of the inputs to Y1 is a fixed default value. The value is specified in the default statement. Default values can be specified for input ports which are also aliased out to an input port of the encompassing module. In this case, the "alias" statements take precedence, and the default values are used only when the input port of the encompassing module is not connected to anything outside.

In this case the data type of the input is a floating point scalar. Floating point scalars are shuttled around in the "omiFloatDataPacket". Other data types include matrices ("omiMatrixDataPacket"), image planes ("omiImagePlaneDataPacket"), and associative arrays of data packets ("omiChannelDataPacket").

Adding new data flow map modules does not require modifying C++ code or re-compiling. Adding new native type C++ modules, new data types, and new indices for associative arrays requires adding C++ code and re-compiling. These changes are highly structured and relatively simple, however they are beyond the scope of this report.

3.4 KNOWN ERRORS AND PLANNED IMPROVEMENTS

3.4.1 KNOWN ERRORS

The algorithm to blend two images using linear combinations on the different visual channel does exactly that. However a non-stationary version is also needed which will achieve the specified blend, but only modify the content interior to the specified target region. The present algorithm is useful for some purposes, but it is not correct for creating low-signature stimuli for perception testing because the changes are not restricted to the target region. At the present time, the background is unchanged only when the coefficients for the linear combinations are identical on all channels. The correction is simple, and was used in the TARDEC Visual Model, but was omitted in VPM.

As described in Section 2, two improvements to the early visual processing model are needed. A model of luminance adaptation and cone non-linearities needs to be incorporated. The current hard-wiring for shape filtering with short, stubby line segments at horizontal and vertical orientations needs to be replaced with a flexible approach that supports a richer (or less rich) set of shape element filters, as appropriate to the application.

3.4.2 PLANNED IMPROVEMENTS

High-level re-organization of the software is planned to improve flexibility, to improve processing speed, and to reduce memory requirements. The current high-level organization wraps many different operations under one operation, Masks. Consequently, changes to the processing flow to produce different metrics requires delving several levels down into the hierarchy of maps. This in turn requires more intimate knowledge of the software organization than a researcher should need to know to obtain different metrics. The current structure is also rife with duplicate computations, and long-term memory allocations.

These problems can be solved at one step by re-organizing the high level application into a set of smaller applications which are then sequenced in a UNIX script. This change will enable researchers using the system to redirect the processing flow for different metrics without modifying the data flow maps.

The modules to read and write binary files are not working. At the present time all file input and output use ASCII files. This is fine and good for data files, so they can be edited easily. However, it is very costly in terms of disk space and processing time for images and multi-resolution image pyramids. The binary file I/O routines need to be fixed. They are not used at the present time in VPL.

The caching scheme presently in the workbench is inadequate. It requires the module builder to decide when and where intermediate results will be cached, and then hard-code this into the data flow maps. This damages the clarity of the data flow maps. The module builders may not always remember to cache at the best points. Once allocated, memory is not released until the application finishes. This creates the possibility that the host machine could run out of memory and crash.

These problems can be corrected by using an automatic chaching scheme that (1) uses up to a specified amount of memory but no more, (2) re-allocates space once the intermediate results has been provided to all the modules that need it, and (3) re-allocates space when the specified memory limit is hit.

APPENDIX A

REFERENCES

Boynton, 1992, *Human Color Vision*, Optical Society of America.

Boynton and Whitten, 1970, "Visual adaptation in monkey cones: recordings of late receptor potentials," *Science*, 170, 1423-1426.

DeValois, et al., 1974, "Psychophysical studies of monkey vision III. Spatial luminance contrast sensitivity tests of macaque and human observers," *Vision Research*, 14, 75-81.

DeValois and DeValois, 1990, *Spatial Vision*, Oxford University Press.

Hubel and Wiesel, 1965, "Receptive fields and functional architecture in two nonstriate visual areas (18 and 19) of the cat," *Journal of Neurophysiology*, 28, 229-289.

Kelly, 1961, "Visual responses to time-dependent stimuli. I. Amplitude sensitivity measurements," *Journal of the Optical Society of America*, 64, 983-990.

LeGrand, 1968, *Light Color and Vision*, Halsted Press.

Rushton and Henry, 1968, "Bleaching and regeneration of cone pigments in man," *Vision Research*, 8, 617-631.

Thompson, 1967, *Foundations of Physiological Psychology*, Harper & Row.

APPENDIX B

CURRENTLY AVAILABLE GENERAL PURPOSE MODULES

B.1 MAP MODULES (~/src/lib/maps)

```
------------------------------------------------------------------------------------------------

./Bandpass/MultiResolution/CircularBandpass/omiCDPMultiResolutionCircularBand
pass.map
   omiCDPMultiResolutionCircularBandpass
      :CDPInput
      :MaskInput
      :MaskInput
      :RemainderCDPOutput
      :CDPOutput

./Bandpass/MultiResolution/CircularBandpass/omiCDPMultiResolutionCircularBand
passInverse.map
   omiCDPMultiResolutionCircularBandpassInverse
      :CDPInput
      :RemainderCDPInput
      :CDPOutput
      :MaskInput

./Bandpass/MultiResolution/MultiResolution2OrientationBandpass/omiCDPMultiReso
lution2OrientationBandpassInverse.map
   omiCDPMultiResolution2OrientationBandpassInverse
      :CDPInput
      :RemainderCDPInput
      :MaskInput
      :CDPOutput

./Bandpass/MultiResolution/MultiResolution2OrientationBandpass/omiIPToCDPMulti
Resolution2OrientationBandpass.map
   omiIPToCDPMultiResolution2OrientationBandpass
      :ImageInput
      :MaskInput
      :IndexInput
      :IndexInput
      :CDPOutput
      :RemainderCDPOutput

./Bandpass/MultiResolution/MultiResolution2OrientationBandpass/omiCDPMultiReso
lution2OrientationBandpass.map
   omiCDPMultiResolution2OrientationBandpass
      :CDPInput
      :MaskInput
      :CDPOutput
      :RemainderCDPOutput

./Bandpass/Orientation/omiCDP2OrientationBandpass.map
   omiCDP2OrientationBP
      :ImageInput
```

```
    :InitInput
    :Output

./Bandpass/Orientation/omiIPDPOrientationBandpassFilter.map
  omiIPDPOrientationBandpassFilter
    :OrientationInput
    :BandwidthInput
    :CenterInput
    :ImageInput
    :Crop.output
    :Output
    :OriginalImageInput
    :ImageInput

./Bandpass/Orientation/omiIPDPToCDP2OrientationBandpass.map
  omiIPDPToCDP2OrientationBandpass
    :ImageInput
    :ImageInput
    :Output

./CDPModules/PosNeg/omiCDPPosNeg.map
  omiCDPPos
    :ImageInput
    :ReOutput
    :ImageInput
    :ImOutput
  omiCDPNeg
    :ImageInput
    :ReOutput
    :ImageInput
    :ImOutput

./CDPModules/omiCDPDivider.map
  omiCDPDiv
    :NumeratorInput
    :DenominatorInput
    :Output

./CDPModules/omiCDPImageClip.map
  omiCDPImageClip
    :Input
    :ClipLoReOutput

./CDPModules/omiCDPImageLog.map
  omiCDPImageLog
    :Input
    :LogBaseInput
    :Output

./CDPModules/omiCDPImageMag.map
  omiCDPImageMag
    :Input
    :Output
```

```
./CDPModules/omiCDPImageStats.map
   omiCDPImageStats
      :Input
      :SumOutput

./CDPModules/omiCDPSign.map
   omiCDPSign
      :Input
      :Output

./CDPModules/omiCDPUniformRandomImage.map
   omiCDPUniformRandomImage
      :ImageInput
      :Output

./CDPModules/omiCDPofFloatAdd.map
   omiCDPofFloatAdd
      :Input
      :OtherInput
      :Output

./CDPModules/omiCDPofFloatDivide.map
   omiCDPofFloatDivide
      :NumeratorInput
      :DenominatorInput
      :Output

./CDPModules/omiCDPofFloatLn.map
   omiCDPofFloatLn
      :Input
      :Output

./CDPModules/omiCDPofFloatMultiply.map
   omiCDPofFloatMultiply
      :Input1
      :Input2
      :Output

./CDPModules/omiCDPofFloatRaiseToPower.map
   omiCDPofFloatRaiseToPower
      :Input_A
      :ExponentInput
      :Output

./CDPModules/omiCDPofFloatSubtract.map
   omiCDPofFloatSubtract
      :PlusInput
      :MinusInput
      :Output

./CDPModules/omiCDPofImageplaneAdder.map
   omiCDPofImageplaneAdder
```

```
        :Input_A
        :Input_B
        :Output

./CDPModules/omiCDPofImageplaneMultiplier.map
    omiCDPofImageplaneMultiplier
        :Input_A
        :Input_B
        :Output

./CDPModules/omiCDPAbs.map
    omiCDPAbs
        :Input
        :Output

./CDPModules/omiCDPofImageplaneRaiseToPower.map
    omiCDPofImageplaneRaiseToPower
        :Input_A
        :PowerInput
        :Output

./CDPModules/omiCDPCollapseToFDP/omiCDPCollapseToFDP.map
    omiCDPCollapseToFDP
        :FloatCDPInput
        :Output

./CDPModules/omiCDPCollapseToIPDP/omiCDPCollapseToIPDP.map
    omiCDPCollapseToIPDP
        :ImageCDPInput
        :Output

./CDPModules/omiCDPofImageplaneSubtract.map
    omiCDPofImageplaneSubtract
        :PlusInput
        :MinusInput
        :Output

./CDPModules/omiCDPofImageplaneDivider.map
    omiCDPofImageplaneDivider
        :NumeratorInput
        :DenominatorInput
        :Output

./CDPModules/omiCDPofFloatMultiplyConstant.map
    omiCDPofFloatMultiplyConstant
        :CDPInput
        :ConstInput
        :Output

./CDPModules/omiCDPFilter.map
    omiCDPFilter
        :ImageInput
        :KernelInput
        :KernelCenter
        :Output

./CDPModules/omiCDPofFloatSubtractConstant.map
    omiCDPofFloatSubtractConstant
        :PlusInput
        :MinusInput
        :Output

./CDPModules/omiCDPofFloatAddConstant.map
    omiCDPofFloatAddConstant
        :CDPInput
        :ConstInput
        :Output

./CDPModules/omiCDPImageSquarer.map
    omiCDPImageSquarer
        :Input
        :Output

./CDPModules/omiCDPThreshold.map
    omiCDPThreshold
        :Input
        :ThresholdValue
        :AboveThresholdVal
        :BelowThresholdVal
        :ThresholdedOutput

./IPMultiResolution2OrientationBandpass/omiIPMultiResolution2OrientationBandpass
.map
    omiIPMultiRes2OrientBp
        :ImageInput
        :MaskInput
        :IndexInput
        :IndexInput
        :BandpassOutput
        :RemainderOutput

./InverseBandpass/omiCDP2OrientInvBandpass.map
    omiCDP2OrientInvBandpass
        :CDPInput
        :CDPInput
        :Output

./InverseBandpass/omiCDPInverseBandpass.map
    omiCDPInverseBandpass
        :CDPInput
        :IndexInput
        :OutputImage

./OrientationBandpass/omi2OrientBp.map
    omi2OrientBp
        :ImageInput
```

```
          :ImageInput
          :ImageInput
          :AllOutput

./OrientationBandpass/omi2OrientationBandpass.map
     omi2OrientBp
          :ImageInput
          :ImageInput
          :ImageInput
          :Output

./OrientationBandpass/omiCDP2OrientationBandpass.map
     omiCDP2OrientationBP
          :ImageInput
          :InitInput
          :Output

./OrientationBandpass/omiIPDPOrientBp.map
     omiIPDPOrientBp
          :OrientationInput
          :BandwidthInput
          :CenterInput
          :ImageInput
          :Crop.output
          :Output
          :OriginalImageInput
          :ImageInput

./OrientationBandpass/omiIPDPOrientationBandpass.map
     omiIPDPOrientBp
          :OrientationInput
          :BandwidthInput
          :CenterInput
          :ImageInput
          :Crop.output
          :Output
          :OriginalImageInput
          :ImageInput

./IPDPModules/PosNeg/omiIPDPNeg.map
     omiIPDPNeg
          :ImageInput
          :ReOutput
          :ImOutput

./IPDPModules/PosNeg/omiIPDPPos.map
     omiIPDPPos
          :ImageInput
          :ReOutput
          :ImOutput


--------------------------------------------------------------------------------
```

B.2 NATIVE TYPE (C++) MODULES (~/src/lib/src)

./omiDataProcessor/DataProcessors/ChannelDataPacketTools/omiCDPAccumulator/omiCDPAccumulator.h
  omiCDPAccumulator
    :CDPInput
    :FunctionInput
    :AccumInput
    :InitInput
    :IndexInput
    :CDPOutput
    :CDPIntermediateResultOutput
    :FunctionOutput
    :StateOutput
    :CurrentOutput
    :IndexOutput

./omiDataProcessor/DataProcessors/ChannelDataPacketTools/omiCDPAdder/omiCDPAdder.c++
  omiCDPAdder
    :Input
    :Output

./omiDataProcessor/DataProcessors/ChannelDataPacketTools/omiCDPAndMask/omiCDPAndMask.c++
  omiCDPAndMask
    :Input
    :Output

./omiDataProcessor/DataProcessors/ChannelDataPacketTools/omiCDPChannelMultiplier/omiCDPChannelMultiplier.h
  omiCDPChannelMultiplier
    :IndexInput
    :CDPInput
    :Output

./omiDataProcessor/DataProcessors/ChannelDataPacketTools/omiCDPCropFFT/omiCDPCropFFT.c++
  omiCDPCropFFT
    :CDPInput
    :DimensionsInput
    :Output

./omiDataProcessor/DataProcessors/ChannelDataPacketTools/omiCDPDivider/omiCDPDivider.c++
  omiCDPDivider
    :NumeratorInput
    :DenominatorInput
    :Output

./omiDataProcessor/DataProcessors/ChannelDataPacketTools/omiCDPExpander/omiCDPExpander.c++
  omiCDPExpander

```
        :MaskInputPortName
        :ImageInputPortName
        :OutputPortName

./omiDataProcessor/DataProcessors/ChannelDataPacketTools/omiCDPFFT/omiCDPF
FT.c++
   omiCDPFFT
        :Input
        :Output

./omiDataProcessor/DataProcessors/ChannelDataPacketTools/omiCDPFFTInverse/om
iCDPFFTInverse.c++
   omiCDPFFTInverse
        :Input
        :Output

./omiDataProcessor/DataProcessors/ChannelDataPacketTools/omiCDPFlatten/omiCD
PFlatten.c++
   omiCDPFlatten
        :Input
        :Output

./omiDataProcessor/DataProcessors/ChannelDataPacketTools/omiCDPFullExpander/
omiCDPFullExpander.h
   omiCDPFullExpander
        :MaskInput
        :ImageInput
        :Output

./omiDataProcessor/DataProcessors/ChannelDataPacketTools/omiCDPIndexJoiner/o
miCDPIndexJoiner.c++
   omiCDPIndexJoiner
        :CDPInput
        :IndexInput
        :Output

./omiDataProcessor/DataProcessors/ChannelDataPacketTools/omiCDPIndexMaker/o
miCDPIndexMaker.c++
   omiCDPIndexMaker
        :CDPInput
        :StartIndexInput
        :EndIndexInput
        :InitialDPInput
        :Output

./omiDataProcessor/DataProcessors/ChannelDataPacketTools/omiCDPIndexStripper/
omiCDPIndexStripper.c++
   omiCDPIndexStripper
        :CDPInput
        :IndexInput
        :Output
```

```
./omiDataProcessor/DataProcessors/ChannelDataPacketTools/omiCDPIndexedAccum
ulator/omiCDPIndexedAccumulator.h
   omiCDPIndexedAccumulator
         :CDPInput
         :FunctionInput
         :AccumInput
         :CIDPIndexInput
         :CDPOutputUp
         :CDPOutputDown
         :FunctionOutput
         :StateOutput
         :CurrentOutput
         :IndexOutput
         :DocOutput

./omiDataProcessor/DataProcessors/ChannelDataPacketTools/omiCDPInsert/omiCDP
Insert.h
   omiCDPInsert
         :PacketToAddInput
         :IndexInput
         :DefaultCDPInput
         :Output

./omiDataProcessor/DataProcessors/ChannelDataPacketTools/omiCDPInvertMask/om
iCDPInvertMask.c++
   omiCDPInvertMask
         :Input
         :Output

./omiDataProcessor/DataProcessors/ChannelDataPacketTools/omiCDPLowpassAnalyz
er/omiCDPLowpassAnalyzer.h
   omiCDPLowpassAnalyzer
         :MaskInput
         :ImageInput
         :Output

./omiDataProcessor/DataProcessors/ChannelDataPacketTools/omiCDPMakeBinary/o
miCDPMakeBinary.c++
   omiCDPMakeBinary
         :Input
         :Output

./omiDataProcessor/DataProcessors/ChannelDataPacketTools/omiCDPMap/omiBinar
yCDPOperator/omiBinaryCDPOperator.h
   omiBinaryCDPOperator
         :CDPInput_A
         :CDPInput_B
         :FunctionInput
         :IndexInput
         :CDPOutput
         :FunctionOutput_A
         :FunctionOutput_B
         :IndexOutput
```

./omiDataProcessor/DataProcessors/ChannelDataPacketTools/omiCDPMap/omiUnary
CDPOperator/omiUnaryCDPOperator.h
  omiUnaryCDPOperator
    :CDPInput
    :FunctionInput
    :IndexInput
    :CDPOutput
    :FunctionOutput
    :IndexOutput

./omiDataProcessor/DataProcessors/ChannelDataPacketTools/omiCDPMax/omiCDPM
ax.c++
  omiCDPMax
    :CDPInput
    :IndexInput
    :Output

./omiDataProcessor/DataProcessors/ChannelDataPacketTools/omiCDPMin/omiCDPMi
n.c++
  omiCDPMin
    :CDPInput
    :IndexInput
    :Output

./omiDataProcessor/DataProcessors/ChannelDataPacketTools/omiCDPMultiplier/omi
CDPMultiplier.c++
  omiCDPMultiplier
    :Input
    :Output

./omiDataProcessor/DataProcessors/ChannelDataPacketTools/omiCDPOrMask/omiC
DPOrMask.c++
  omiCDPOrMask
    :Input
    :Output

./omiDataProcessor/DataProcessors/ChannelDataPacketTools/omiCDPSelect/omiCDP
Select.h
  omiCDPSelect
    :CDPInput
    :IndexInput
    :Output

./omiDataProcessor/DataProcessors/ChannelDataPacketTools/omiCDPSetDifference/o
miCDPSetDifference.h
  omiCDPSetDifference
    :PlusInput
    :MinusInput
    :Output

./omiDataProcessor/DataProcessors/ChannelDataPacketTools/omiCDPSingletonStripp
er/omiCDPSingletonStripper.c++

```
   omiCDPSingletonStripper
        :CDPInput
        :Output

./omiDataProcessor/DataProcessors/ChannelDataPacketTools/omiCDPSplit/omiCDPS
plit.h
   omiCDPSplit
        :CDPInput
        :IndexInput
        :Output

./omiDataProcessor/DataProcessors/ChannelDataPacketTools/omiCDPSubtractor/omi
CDPSubtractor.c++
   omiCDPSubtractor
        :PlusInput
        :MinusInput
        :Output

./omiDataProcessor/DataProcessors/ChannelDataPacketTools/omiCDPUnion/omiCDP
Union.c++
   omiCDPUnion
        :Input
        :Output

./omiDataProcessor/DataProcessors/ChannelDataPacketTools/omiCDPDuplicateStruc
ture/omiCDPStructureDuplicator.h
   omiCDPStructureDuplicator
        :CDPTemplateInput
        :CDPDataInput
        :CDPOutput

./omiDataProcessor/DataProcessors/ChannelDataPacketTools/omiCDPNormalRandom
Image/omiCDPNormalRandomImage.h
   omiCDPNormalRandomImage
        :CDPInput
        :SeedInput
        :MeanInput
        :SigmaInput
        :CDPOutput

./omiDataProcessor/DataProcessors/ChannelIndexDataPacketTools/omiCIDPValueExt
ractor/omiCIDPValueExtractor.h
   omiCIDPValueExtractor
        :CIDPInput
        :WhichCIDPValueToExtractInput
        :FDPOutput

./omiDataProcessor/DataProcessors/DataPacketTools/omiAsciiFileInput/omiAsciiFileI
nput.c++
   omiAsciiFileInput
        :OutputPortName
```

```
./omiDataProcessor/DataProcessors/DataPacketTools/omiAsciiFileOutput/omiAsciiFil
eOutput.c++
   omiAsciiFileOutput
         :InputPortName

./omiDataProcessor/DataProcessors/DataPacketTools/omiBinaryFileInput/omiBinaryF
ileInput.c++
   omiBinaryFileInput
         :OutputPortName

./omiDataProcessor/DataProcessors/DataPacketTools/omiBinaryFileOutput/omiBinar
yFileOutput.c++
   omiBinaryFileOutput
         :InputPortName

./omiDataProcessor/DataProcessors/DataPacketTools/omiDPMultiplexer/omiDPMultip
lexer.c++
   omiDPMultiplexer
         :Input
         :SelectInput
         :SelectOnInput
         :Output

./omiDataProcessor/DataProcessors/DataPacketTools/omiImmediateAsciiFileInput/om
iImmediateAsciiFileInput.c++
   omiImmediateAsciiFileInput
         (port names depend on file contents)

./omiDataProcessor/DataProcessors/DataPacketTools/omiRgbFileInput/omiRgbFileInp
ut.c++
   omiRgbFileInput
         :OutputPortName

./omiDataProcessor/DataProcessors/DataPacketTools/omiSimpleCache/omiDPSimple
Cache.h
   omiDPSimpleCache
         :Input
         :Output

./omiDataProcessor/DataProcessors/FloatDataPacketTools/omiFDPExp/omiFDPExp.h
   omiFDPExp
         :Input
         :Output

./omiDataProcessor/DataProcessors/FloatDataPacketTools/omiFDPLn/omiFDPLn.h
   omiFDPLn
         :Input
         :Output

./omiDataProcessor/DataProcessors/FloatDataPacketTools/omiFDPPower/omiFDPPow
er.h
   omiFDPPower
         :Input
```

```
        :ExponentInput
        :Output

./omiDataProcessor/DataProcessors/FloatDataPacketTools/omiFloatDataPacketAdder/
omiFloatDataPacketAdder.h
   omiFloatDataPacketAdder
        :Input
        :Output

./omiDataProcessor/DataProcessors/FloatDataPacketTools/omiFloatDataPacketDivider
/omiFloatDataPacketDivider.c++
   omiFloatDataPacketDivider
        :NumeratorInput
        :DenominatorInput
        :Output

./omiDataProcessor/DataProcessors/FloatDataPacketTools/omiFloatDataPacketInput/
omiFloatDataPacketInput.c++
   omiFloatDataPacketInput
        :output

./omiDataProcessor/DataProcessors/FloatDataPacketTools/omiFloatDataPacketMultipl
ier/omiFloatDataPacketMultiplier.h
   omiFloatDataPacketMultiplier
        :Input
        :Output

./omiDataProcessor/DataProcessors/FloatDataPacketTools/omiFloatDataPacketOutpu
t/omiFloatDataPacketOutput.c++
   omiFloatDataPacketOutput
        :input

./omiDataProcessor/DataProcessors/FloatDataPacketTools/omiFloatDataPacketSqrt/o
miFloatDataPacketSqrt.c++
   omiFloatDataPacketSqrt
        :Input
        :Output

./omiDataProcessor/DataProcessors/FloatDataPacketTools/omiFloatDataPacketSubtra
ctor/omiFloatDataPacketSubtractor.c++
   omiFloatDataPacketSubtractor
        :PlusInput
        :MinusInput
        :Output

./omiDataProcessor/DataProcessors/FloatDataPacketTools/omiFDPGenerateEyeNoise
Magnitude/omiFDPGenerateEyeNoiseMagnitude.c++
   omiFDPGenerateEyeNoiseMagnitude
        :CIDPInput
        :Output

./omiDataProcessor/DataProcessors/FloatDataPacketTools/omiFloatDataPacketAbs/o
miFloatDataPacketAbs.c++
  omiFloatDataPacketAbs
        :Input
        :Output

./omiDataProcessor/DataProcessors/FloatDataPacketTools/omiFloatDataPacketSign/o
miFloatDataPacketSign.c++
  omiFloatDataPacketSign
        :Input
        :Output

./omiDataProcessor/DataProcessors/ImagePlaneDataPacketTools/omiIPDPAbs/omiIPD
PAbs.h
  omiIPDPAbs
        :ImageInput
        :ReOutput
        :ImOutput

./omiDataProcessor/DataProcessors/ImagePlaneDataPacketTools/omiIPDPAdder/omiI
PDPAdder.h
  omiIPDPAdder
        :Input
        :Output

./omiDataProcessor/DataProcessors/ImagePlaneDataPacketTools/omiIPDPAndMask/o
miIPDPAndMask.c++
  omiIPDPAndMask
        :Input
        :Output

./omiDataProcessor/DataProcessors/ImagePlaneDataPacketTools/omiIPDPClip/omiIP
DPClip.h
  omiIPDPClip
        :ImageInput
        :ClipValueInput
        :ClipHiReOutput
        :ClipHiImOutput
        :ClipLoReOutput
        :ClipLoImOutput

./omiDataProcessor/DataProcessors/ImagePlaneDataPacketTools/omiIPDPConvolve/o
miIPDPConvolve.c++
  omiIPDPConvolve
        :ImageInput
        :MaskInput
        :CoordinatesInput
        :Output

./omiDataProcessor/DataProcessors/ImagePlaneDataPacketTools/omiIPDPCropFFT/o
miIPDPCropFFT.h
  omiIPDPCropFFT
        :ImageInput
        :DimensionsInput
        :Output
```

```
./omiDataProcessor/DataProcessors/ImagePlaneDataPacketTools/omiIPDPDivider/omi
IPDPDivider.c++
   omiIPDPDivider
        :NumeratorInput
        :DenominatorInput
        :Output

./omiDataProcessor/DataProcessors/ImagePlaneDataPacketTools/omiIPDPExclude/om
iIPDPExclude.c++
   omiIPDPExclude
        :ImageInput
        :ValueInput
        :EpsilonInput
        :Output

./omiDataProcessor/DataProcessors/ImagePlaneDataPacketTools/omiIPDPExpander/o
miIPDPExpander.h
   omiIPDPExpander
        :ImageInput
        :MaskInput
        :SizeImageInput
        :Output

./omiDataProcessor/DataProcessors/ImagePlaneDataPacketTools/omiIPDPFFT/omiIP
DPFFT.h
   omiIPDPFFT
        :ImageInput
        :Output
        :CenterOutput

./omiDataProcessor/DataProcessors/ImagePlaneDataPacketTools/omiIPDPFFTInverse/
omiIPDPFFTInverse.h
   omiIPDPFFTInverse
        :ImageInput
        :Output

./omiDataProcessor/DataProcessors/ImagePlaneDataPacketTools/omiIPDPInvertMask
/omiIPDPInvertMask.c++
   omiIPDPInvertMask
        :ImageInput
        :Output

./omiDataProcessor/DataProcessors/ImagePlaneDataPacketTools/omiIPDPLog/omiIPD
PLog.h
   omiIPDPLog
        :ImageInput
        :LogBaseInput
        :Output

./omiDataProcessor/DataProcessors/ImagePlaneDataPacketTools/omiIPDPLowpassAn
alyzer/omiIPDPLowpassAnalyzer.h
   omiIPDPLowpassAnalyzer
```

```
        :ImageInput
        :MaskInput
        :Output

./omiDataProcessor/DataProcessors/ImagePlaneDataPacketTools/omiIPDPMag/omiIP
DPMag.c++
   omiIPDPMag
        :ImageInput
        :Output

./omiDataProcessor/DataProcessors/ImagePlaneDataPacketTools/omiIPDPMakeBinary
/omiIPDPMakeBinary.c++
   omiIPDPMakeBinary
        :ImageInput
        :Output

./omiDataProcessor/DataProcessors/ImagePlaneDataPacketTools/omiIPDPMaskMaker
/omiIPDPMaskMaker.c++
   omiIPDPMaskMaker
        :RegionInput
        :ImageInput
        :PixelValuesInput
        :Output

./omiDataProcessor/DataProcessors/ImagePlaneDataPacketTools/omiIPDPMultiplier/o
miIPDPMultiplier.c++
   omiIPDPMultiplier
        :Input
        :Output

./omiDataProcessor/DataProcessors/ImagePlaneDataPacketTools/omiIPDPOrMask/om
iIPDPOrMask.c++
   omiIPDPOrMask
        :Input
        :Output

./omiDataProcessor/DataProcessors/ImagePlaneDataPacketTools/omiIPDPOrientation
Bandpass/omiIPDPOrientationBandpass.c++
   omiIPDPOrientationBandpass
        :ImageInput
        :CenterInput
        :OrientationInput
        :BandwidthInput
        :Output

./omiDataProcessor/DataProcessors/ImagePlaneDataPacketTools/omiIPDPRaiseToPow
er/omiIPDPRaiseToPower.c++
   omiIPDPRaiseToPower
        :ImageInput
        :PowerInput
        :Output
```

```
./omiDataProcessor/DataProcessors/ImagePlaneDataPacketTools/omiIPDPRandomIma
ge/NormalRandomImage/omiIPDPNormalRandomImage.h
   omiIPDPNormalRandomImage
        :ImageInput
        :SeedValueInput
        :MeanInput
        :SigmaInput
        :Output

./omiDataProcessor/DataProcessors/ImagePlaneDataPacketTools/omiIPDPRandomIma
ge/UniformRandomImage/omiIPDPUniformRandomImage.h
   omiIPDPUniformRandomImage
        :ImageInput
        :SeedValueInput
        :LowerBound
        :UpperBound
        :Output

./omiDataProcessor/DataProcessors/ImagePlaneDataPacketTools/omiIPDPReSample/o
miIPDPReSample.h
   omiIPDPReSample
        :ImageInput
        :StartInput
        :EndInput
        :NewSizeInput
        :Output

./omiDataProcessor/DataProcessors/ImagePlaneDataPacketTools/omiIPDPRealPart/o
miIPDPRealPart.c++
   omiIPDPRealPart
        :ImageInput
        :Output

./omiDataProcessor/DataProcessors/ImagePlaneDataPacketTools/omiIPDPSign/omiIP
DPSign.h
   omiIPDPSign
        :Input
        :Output

./omiDataProcessor/DataProcessors/ImagePlaneDataPacketTools/omiIPDPStats/omiIP
DPStats.h
   omiIPDPStats
        :ImageInput
        :MaskInput
        :MeanOutput
        :CountOutput
        :SumOutput
        :VarianceOutput
        :StdDevOutput

./omiDataProcessor/DataProcessors/ImagePlaneDataPacketTools/omiIPDPSubtracter/
omiIPDPSubtracter.c++
   omiIPDPSubtracter
```

```
        :PlusInput
        :MinusInput
        :Output

./omiDataProcessor/DataProcessors/ImagePlaneDataPacketTools/omiIPDPThresholder
/omiIPDPThresholder.h
   omiIPDPThresholder
        :ImageInput
        :ThresholdValueInput
        :GreaterThanThresholdValue
        :LessThanThresholdValue
        :ReOutput
        :ImOutput

./omiDataProcessor/DataProcessors/ImagePlaneDataPacketTools/omiIPDPUniformIma
ge/omiIPDPUniformImage.c++
   omiIPDPUniformImage
        :CoordinatesInput
        :ValueInput
        :Output

./omiDataProcessor/DataProcessors/ImagePlaneDataPacketTools/omiIPDPMakeRectan
gle/omiIPDPMakeRectangle.h
   omiIPDPMakeRectangle
        :ImageInput
        :RectangleCenterInput
        :RectangleSizeInput
        :InsideValueInput
        :OutsideValueInput
        :Output

./omiDataProcessor/DataProcessors/ImagePlaneDataPacketTools/omiIPDPSizeToMDP
/omiIPDPSizeToMDP.h
   omiIPDPSizeToMDP
        :ImageInput
        :Output

./omiDataProcessor/DataProcessors/ImagePlaneDataPacketTools/omiIPDPSquarer/om
iIPDPSquarer.h
   omiIPDPSquarer
        :IPDPInput
        :IPDPOutput

./omiDataProcessor/DataProcessors/MatrixDataPacketTools/omiMatrixDataPacketAdd
er/omiMatrixDataPacketAdder.c++
   omiMatrixDataPacketAdder
        :Input
        :Output

./omiDataProcessor/DataProcessors/MatrixDataPacketTools/omiMatrixDataPacketDivi
der/omiMatrixDataPacketDivider.c++
   omiMatrixDataPacketDivider
        :NumeratorInput
        :DenominatorInput
        :Output

./omiDataProcessor/DataProcessors/MatrixDataPacketTools/omiMatrixDataPacketMul
tiplier/omiMatrixDataPacketMultiplier.c++
   omiMatrixDataPacketMultiplier
        :Input
        :Output

./omiDataProcessor/DataProcessors/MatrixDataPacketTools/omiMatrixDataPacketSub
tractor/omiMatrixDataPacketSubtractor.c++
   omiMatrixDataPacketSubtractor
        :PlusInput
        :MinusInput
        :Output

./omiDataProcessor/DataProcessors/RegionDataPacketTools/omiRegionDataPacketMa
sk/omiRegionDataPacketMask.c++
   omiRegionDataPacketMask
        :ImageInput
        :RegionInput
        :Output
```

APPENDIX C

APPLICATION-SPECIFIC MODULES IN NAC VPM

C.1 MAP MODULES (~/src/nac-vpm/maps)

```
-----------------------------------------------------------------------------

./Adapt/omiAdapt2.map
  omiAdapt2
        :SearchImagePowerPyramid
        :XYZMaskPyramid
        :X_Input
        :Y_Input
        :Z_Input
        :W_Input
        :EyeNoisePower
        :BkgPower
        :AdaptiveGains

./BandPfromLowP/omiBPfromLP.map
  omiBPfromLP
        :KernelInput
        :MaskLPInput
        :KernelInput
        :ImageLPInput
        :DividedInput
        :ImageLPInput
        :Output
        :RemainderOutput

./BlendOut/omiBlendOut2.map
  omiBlendOut2
        :ImageInput
        :RegionInput
        :PixelValuesInput
        :KernelInput
        :IPMaskOutput
        :OutputRGBImage

./BlendOut/omiBlendOut.map
  omiBlendOut
        :ImageInput
        :RegionInput
        :PixelValuesInput
        :KernelInput
        :KernelInput
        :IPMaskOutput
        :OutputRGBImage

./CDF/omiCDPCDF.map
  omiCDPCDF
        :Input
        :N50Input
        :ExponentInput
        :Output

./CDF/omiCDPImageCDF.map
   omiCDPImageCDF
        :Input
        :N50Input
        :GammaInput
        :Output

./CDF/omiPooling.map
   omiPooling
        :Input
        :ExponentInput
        :ExponentInput
        :Output

./CDF/omiFDPCDF.map
   omiFDPCDF
        :Input
        :N50Input
        :ExponentInput
        :Output

./CDF/omiNthRootofSumofNthPower.map
   omiNthRootofSumofNthPower
        :Input
        :ExponentInput
        :WeightInput
        :ExponentInput
        :Output

./CDF/omiFloatNormalCDF.map
   omiFloatNormalCDF
        :Input
        :N50
        :Gamma
        :Output

./CDF/omiWeightedPooling.map
   omiWeightedPooling
        :Input
        :ExponentInput
        :WeightInput
        :ExponentInput
        :Output

./EarlyVisionAnalyzer/omiEarlyVisionAnalyzer.map
   omiEarlyVisionAnalyzer
        :rgb_channel_input
        :MaskInput
        :RemainderCDPOutput
        :COAOutput
        :internal_noise_output
        :CDPOutput
        :bw_mean_output

./EarlyVisionAnalyzer/omiEarlyVisionAnalyzerInverse.map
   omiEarlyVisionAnalyzerInverse
        :MaskInput
        :CDPInput
        :RemainderCDPInput
        :CDPOutput

./EdgeDetector/omiCDPEdgeDetector.map
   omiCDPEdgeDetector
        :Input
        :Output

./Excluder/omiCDPExclude.map
   omiCDPExclude
        :Input
        :ExcludedOutput
        :EpsilonInput
        :ExcludeTargetValue

./GenerateFreqPlaneWeights/omiGenerateFreqPlaneWeights.map
   omiGenerateFreqPlaneWeights
        :Input
        :Output
        :CDPInput
        :CDPOfWeightsOutput
        :CDPOfFloatsOutput

./IndexMakers/omi2DindexMaker.map
   omi2DindexMaker
        :InitialDPInput
        :CDPInput
        :Output

./IndexMakers/omiRGB_2DindexMaker.map
   omiRGB
        :InitialDPInput
        :CDPInput
        :Output

./IndexMakers/omi4indexMaker.map
   omi4indexMaker
        :InitialDPInput
        :CDPInput
        :Output

./IndexMakers/omiColOppindexMaker.map
   omiColOppindexMaker
        :InitialDPInput
        :CDPInput
        :Output

./IndexMakers/omiFreqindexMaker.map
   omiFreqindexMaker
        :InitialDPInput
        :CDPInput
        :Output

./IndexMakers/omiRGBindexMaker.map
   omiRGBindexMaker
        :InitialDPInput
        :CDPInput
        :Output

./MakeBackgroundImage/omiMakeBkgrdImage.map
   omiMakeBkgrdImage
        :TMaskInput
        :NTImageInput
        :SeedValueInput
        :MeanInput
        :SigmaInput
        :NTImageInput
        :KernelInput
        :KernelInput
        :KernelInput
        :KernelInput
        :OutputImage
        :RegionMaskPyramidOutput
        :PerimeterMaskPyramidOutput
        :OuterPerimeterMaskPyramidOutput

./MakeMultiresolutionPyramids/omiMakeMultiresolutionPyramids.map
   omiMakeMultiresolutionPyramids
        :KernelInput
        :KernelInput
        :InvTargetMaskInput
        :NoTargetMaskInput
        :SearchMaskInput
        :SearchImageInput
        :NoTargetImageInput
        :InvTargetMaskPyramidOutput
        :NoTargetMaskPyramidOutput
        :SearchMaskPyramidOutput
        :NoTargetImagePowerPyramidOutput
        :EVASearchImageOutput
        :EVANoTargetImageOutput
        :DeltaTargetPowerPyramidOutput
        :SearchImagePowerPyramidOutput
        :FrequencyPlaneWeightsOutput

./MaskMakers/omiRGBMaskMaker.map
   omiRGBMaskMaker
        :ImageInput
        :RegionInput
        :PixelValuesInput
        :Output
        :BareMaskOutput

./MaskMixer/omiMaskMixer.map
   omiMaskMixer
        :Image1Input
        :Image2Input
        :KernelInput
        :KernelInput
        :Image1Input
        :Image2Input
        :MaskInputPlane
        :MaskInputPlane
        :Alpha
        :Image_B_Out
        :ImageOutput

./Mixer/omiMixer.map
   omiMixer
        :Image1Input
        :Image2Input
        :MaskInput
        :MaskInput
        :FreqWeights
        :OrientationWeights
        :ColorOpponentWeights
        :OutputImage

./PreProcessing/omiPreProcessing.map
   omiPreProcessing
        :NoTargetImage
        :SearchImage
        :TargetMask
        :SearchMask
        :InvTgtMask
        :NoTgtMask
        :OriginalImageInput
        :SearchRegion
        :TargetRegion
        :KernelInput
        :KernelInput

./PyramidAnalyzer/omiPyramidAnalyzer.map
   omiPyramidAnalyzer
        :rgb_channel_input
        :kernel_input
        :init_input
        :mask_input
        :VarianceOutput
        :CDPOutput
```

```
./Spotlighter/omiSpotlighter.map
  omiSpotlighter
        :ImageInput
        :MaskInput
        :scalarN
        :CDPOutput

./Stats/CountStats/omiCDPCountStats.map
  omiCDPCountStats
        :Input
        :MaskInput
        :CountOutput

./Stats/MeanStats/omiCDPMeanStats.map
  omiCDPMeanStats
        :Input
        :MaskInput
        :MeanOutput

./Stats/StdDevStats/omiCDPStdDevStats.map
  omiCDPStdDevStats
        :Input
        :MaskInput
        :StdDevOutput

./Stats/StdDevStats/omiCDPStdDevStatsNoMask.map
  omiCDPStdDevStatsNoMask
        :Input
        :StdDevOutput

./Stats/SumStats/omiCDPSumStats.map
  omiCDPSumStats
        :Input
        :MaskInput
        :SumOutput

./Stats/VarianceStats/omiCDPVarianceStats.map
  omiCDPVarianceStats
        :Input
        :MaskInput
        :VarianceOutput

./Stats/RMSStats/omiCDPRMSStats.map
  omiCDPRMSStats
        :Input
        :MaskInput
        :Output

./TargetMaskAnalyzer/omiMaskAnalyzer.map
  omiMaskAnalyzer
        :Input
        :Output
```

```
./Threshholder/omiCDPMultiResolutionThreshold.map
  omiCDPMultiResolutionThreshold
       :Input
       :ThresholdValue
       :ThresholdedOutput
       :AboveThresholdVal
       :BelowThresholdVal

./WeightExpandSum/omiWeightExpandSum2.map
  omiWeightExpandSum2
       :WeightCDPInput
       :MaskInput
       :RemainderCDPInput
       :PoolingT
       :ImageCDPInput
       :PoolingT
       :CDPOutput

./BoundaryProcessor/omiBoundaryProcessor.map
  omiBoundaryProcessor
       :InverseTargetMaskInput
       :ImageInput
       :MaskInput
       :GammaInput
       :N50Input
       :MaskInput
       :WeightsInput
       :EdgeMaskPyramidOutput
       :MREdgeImageOutput

./MakeNACPyramids/omiMakeNACPyramids.map
  omiMakeNACPyramids
       :SceneImageInput
       :SearchRegionInput
       :TargetRegionInput
       :KernelInput
       :KernelInput
       :KernelInput
       :NoTargetImageOutput
       :NoTargetMaskOutput
       :InvTargetMaskOutput
       :SearchImageOutput
       :SearchMaskOutput
       :DeltaTargetPowerPyramidOutput
       :SearchImagePowerPyramidOutput
       :NoTargetImagePowerPyramidOutput
       :InvTargetMaskPyramidOutput
       :NoTargetMaskPyramidOutput
       :SearchMaskPyramidOutput
       :EVASearchImageOutput
       :EVANoTargetImageOutput
       :FrequencyPlaneWeightsOutput
       :CamouflagedImageOutput
```

```
        :RegionMaskPyramidOutput
        :PerimeterMaskPyramidOutput
        :OuterPerimeterMaskPyramidOutput
        :BkgPowerOutput
        :DeltaTexturePowerOutput
        :BkgTextureOutput

./Resample/omiCDPResample.map
  omiCDPResample
        :MDPNewSizeInput
        :MDPNewToCurrentIFOVScaleInput
        :MDPCenterInput
        :CDPInput
        :CDPOutput

./Resample/omiResample.map
  omiResample
        :MDPNewImageSizeInput
        :MDPNewToCurrentIFOVScaleInput
        :MDPCenterInput
        :MDPCenterInput
        :LowerRightMDPOutput
        :UpperLeftMDPOutput
        :IPDPInput
        :MDPNewSizeInput
        :MDPNewToCurrentIFOVScaleInput
        :MDPCenterInput
        :MDPNewSizeInput
        :IPDPInput
        :IPDPOutput

./PredProb/omiPredProbCalc.map
  omiPredProbCalc
        :Content
        :Mask
        :Mask
        :Kernel
        :FrequencyWeights
        :AdaptGains
        :N50A1
        :GA1
        :N50A2
        :GA2
        :N50C1
        :GC1
        :PoolingT
        :Image_Output
        :PredProbA
        :PredProbC1
        :PredProbMetricC1
        :PredProbMetricA
        :Content
        :MaskPyramid
        :AdaptGains
        :AdaptGains
        :AdaptGains
        :N50B1
        :GB1
        :N50C2
        :GC2
        :PoolingT
        :PoolingT
        :PredProbB
        :PredProbC2
        :PredProbMetricC2
```

C.2 NATIVE TYPE (C++) MODULES (~/src/nac-vpm/src)

```
./Path/to/file.ext relative to ~/src/nac-vpm/src
   name-of-module (Used as "instance name-of-module xyz")
      list-of-ports
```

This file was last updated February 1996.

```
./CDF/omiIPDPCDF.h
   omiIPDPCDF
        :ImageInput
        :GammaInput
        :N50Input
        :Output

./EarlyVisionAnalyzer/ColorOpponentAnalyzer/omiColorOpponentAnalyzer.c++
   omiColorOpponentAnalyzer
        :rgb_channel_input
        :xyz_to_sml_input
        :rgb_to_xyz_input
        :sml_to_coloropp_input
        :rgb_bias_input
        :gamma_correction_input
        :gamma_correction_gain_input
        :channel_output
        :bw_mean_output

./EarlyVisionAnalyzer/ColorOpponentAnalyzerInverse/omiColorOpponentAnalyzerInver
se.c++
   omiColorOpponentAnalyzerInverse
        :color_opponent_channel_input
        :xyz_to_sml_input
        :rgb_to_xyz_input
        :sml_to_coloropp_input
        :rgb_bias_input
        :gamma_correction_input
        :gamma_correction_gain_input
        :channel_output

./EarlyVisionAnalyzer/InternalNoiseAnalyzer/omiInternalNoiseAnalyzer.c++
   omiInternalNoiseAnalyzer
        :template_input
        :ifov_input
        :contrast_sens_gain_input
        :temporal_frequency_contrast_sensitivity_gain_input
        :contrast_sens_freq_cutoff_input
        :contrast_sens_rolloff_input
        :signal_exponent_input
        :luminance_input
        :bw_mean_input
        :internal_noise_output

./EdgeDetector/omiIPDPEdgeDetector.c++
   omiIPDPEdgeDetector
        :Input
        :Output

./EyeNoiseMagnitude/omiEyeNoiseMagnitude.c++
   omiEyeNoiseMagnitude
        :B_Input
        :F_Input
        :IFOV_Input
        :A_Input
        :C50_Input
        :T_Input
        :Fb_Input
        :Input
        :eye_noise_magnitude_output

./NormalCDF/omiIPDPNormalCDF.h
   omiIPDPNormalCDF
        :ImageInput
        :GammaInput
        :N50Input
        :Output
```